NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN, IMPLEMENTATION, AND TESTING OF A HIGH PERFORMANCE SUMMATION ADDER FOR RADAR IMAGE SYNTHESIS

by

Major Craig A. Amundson

September 2001

Thesis Advisor: Douglas Fouts
Co-Advisor: Phillip Pace

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** September 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE Design, Implementation, and Testing of A High Performance Summation Adder for Radar Image Synthesis		**5. FUNDING NUMBERS**
6. AUTHOR (S) Craig A. Amundson		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School** Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) **Office of Naval Research**		**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the U.S. Department of Defense or the U.S. Government.		
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.		**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

This thesis documents the schematic design, simulation, and fabrication mask layout of a high-speed 16-bit summation adder to be integrated into the Digital Image Synthesizer (DIS) Application Specific Integrated Circuit (ASIC). The DIS is a single-chip false target radar image generator to be used in countering wide band imaging radars. The DIS will calculate the false target image in 512 range bins. Each range bit utilizes two identical 16-bit binary adders. The 16-Bit Adder must compute the sum of two 16-bit numbers, providing a 16-bit sum, carry output, and overflow detection bit. The stated goal is for this adder to perform all of these functions in one pipeline stage while being clocked at 600 MHz.

The first part of the design process includes an extensive analysis to utilize the fewest gates in designing the simplest adder that can achieve the 600 MHz goal. SPICE net lists are extracted from these schematic designs and simulations conducted to verify logic functionality and propagation speed. Mask layout of the verified design is constructed using a CMOS 0.18 micron process utilizing deep sub-micron technology with six metal interconnect layers. The mask layout design is verified by ensuring all design rule checks (DRC) and layout versus schematic (LVS) checks are satisfied. In aAddition, conclusions and recommendations are provided to assist other DIS project members in using this adder and the aforementioned design process for additional components of the DIS ASIC.

14. SUBJECT TERMS VLSI, ASIC, CMOS, Adder, Chip Design			**15. NUMBER OF PAGES**
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

NSN 7540-01-280-5500 **Standard Form 298 (Rev. 2-89) Prescribed by ANSI Std. 239-18**

THIS PAGE INTENTIONALLY LEFT BLANK

DESIGN, IMPLEMENTATION, AND TESTING OF A HIGH PERFORMANCE SUMMATION ADDER FOR RADAR IMAGE SYNTHESIS

Craig A. Amundson
Major, United States Marine Corps
B.S., Iowa State University, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 2001

Author: [illegible]
Craig A. Amundson

Approved by: Douglas J. Fouts
Prof Douglas Fouts, Thesis Advisor

[illegible]
Prof Phillip Pace, Co-Advisor

[illegible]
Prof Jeffrey Knorr, Chairman
Department of Electrical Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis documents the schematic design, simulation, and fabrication mask layout of a high-peed 16-bit summation adder to be integrated into the Digital Image Synthesizer (DIS) Application Specific Integrated Circuit (ASIC). The DIS is a single-chip false target radar image generator to be used in countering wide band imaging radars. The DIS will calculate the false target image in 512 range bins. Each range bit utilizes two identical 16-bit binary adders. The 16-Bit Adder must compute the sum of two 16-bit numbers, providing a 16-bit sum, carry output, and overflow detection bit. The stated goal is for this adder to perform all of these functions in one pipeline stage while being clocked at 600 MHz.

The first part of the design process includes an extensive analysis to utilize the fewest gates in designing the simplest adder that can achieve the 600 MHz goal. SPICE net lists are extracted from these schematic designs and simulations conducted to verify logic functionality and propagation speed. Mask layout of the verified design is constructed using a CMOS 0.18 micron process utilizing deep sub-micron technology with six metal interconnect layers. The mask layout design is verified by ensuring all design rule checks (DRC) and layout versus schematic (LVS) checks are satisfied. In addition, conclusions and recommendations are provided to assist other DIS project members in using this adder and the aforementioned design process for additional components of the DIS ASIC.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

The author would like to extend thanks and appreciation to his thesis advisors, Professors Fouts and Pace, whose technical proficiency is only exceeded by their love for God. “So whether you eat or drink or whatever you do, do it all for the glory of God.” (1Cor 10:31, NIV)

This work was supported in part by the office of Naval Research Code ONR-313 and the Naval Research Laboratory.

The author thanks God for the support of an incredible wife, Shelly, and two beautiful daughters, Sarah and Katherine.

Finally, the author would ask all persons consider “That if you confess with your mouth, ‘Jesus is Lord,’ and believe in your heart that God raised him from the dead you will be saved.” (Rm 10:9, NIV)

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

This thesis documents the schematic design, simulation, and fabrication mask layout of a high-speed 16-bit summation adder to be integrated into the Digital Image Synthesizer (DIS) Application Specific Integrated Circuit (ASIC). The DIS is a single-chip false target radar image generator to be used in countering wide band imaging radars. The DIS will calculate the false target image in 512 range bins. Each range bit utilizes two identical 16-bit binary adders. The 16-Bit Adder must compute the sum of two 16-bit numbers, providing a 16-bit sum, carry output, and overflow detection bit. The stated goal is for this adder to perform all of these functions in one pipeline stage while being clocked at 600 MHz.

The first part of the design process included an extensive analysis to utilize the fewest gates in designing the simplest adder that can achieve the 600 MHz goal. SPICE net lists were extracted from these schematic designs and simulations conducted to verify logic functionality and propagation speed. In sequence the following designs were explored:

1. 16 x 1-Bit Arithmetic Logic Units (ALU) w/Ripple Carry
2. 4 x 4-Bit Medium Scale Integrated (MSI) Adders w/Ripple Carry
3. 4 x 4-Bit Pseudo-NMOS NOR Logic Adders w/Ripple Carry
4. 4 x 4-Bit CMOS NAND Logic Adders w/Ripple Carry

5. 4 x 4-Bit CMOS NAND Logic Adders w/Carry Look-Ahead

It was necessary to adopt the carry look-ahead technique in order to meet the stated goal. This schematic design underwent extensive SPICE simulation to include being pipelined with registers to verify that it would function within the DIS architecture.

The fabrication mask layout of the verified design was constructed using a CMOS 0.18 micron process utilizing deep sub-micron technology with six metal interconnect layers. The mask layout design was verified by ensuring all design rule checks (DRC) and layout versus schematic (LVS) checks were satisfied. The result of this research is a fabrication mask layout for a 16-bit adder that accomplishes the stated goal.

In addition, a concluding summary and recommendations for future work are provided to assist other DIS project members in using this adder and the aforementioned design process for additional components of the DIS ASIC.

I. INTRODUCTION

A. DIGITAL IMAGE SYNTHESIZER (DIS)

The inverse synthetic aperture radar (ISAR) technique is a well-known method of constructing three-dimensional (3-D) range-Doppler images of moving targets such as ships and aircraft. The United States Navy utilizes this technique with it's AN/APS-137B(V)5 radar, currently deployed on P-3C aircraft [1]. ISAR provides detection, classification and tracking capability against surface and surfaced submarine targets. It also provides range, bearing and positional data on all selected targets, plus medium- or high-resolution images for display and recording.

Other countries are developing implementations of this technology. The Russian Federation and Associated States (CIS) has developed the Sea Dragon maritime surveillance mission system architecture, which utilizes the ISAR technique. It is reported that they are deploying, or are proposing to deploy, this system on their Ilyushin Il-38 ('May') and Tupolev Tu-204P maritime patrol aircraft and Tupolev Tu142M-Z ('Bear-F' Mod 4) Anti-Submarine Warfare (ASW) along with India's fleet of eight Tu-142MK-E ('Bear' Mod 3), five Il-38 and 14 Ka-28 ('Helix-A') ASW platforms [2].

In the future, it is expected that missiles will utilize ISAR in their terminal homing phase in order to reject decoys and increase aim point accuracy, resulting in a greater probability of kill [3]. It is likely that the current suite of counter-homing techniques, electronic attack, distraction chaff and seduction chaff, as well as decoy repeaters will not be practical against ISAR seekers [4].

Consequently, the present need for a counter-targeting solution and the future need for a counter-homing solution against ISAR remain a high-priority for many electronic warfare (EW) systems. A practical method of generating a false target to counter an ISAR system is not available from traditional acoustic charge transport (ACT) and optical devices [5,6]. However, recent advances in application specific integrated circuits (ASICs) have greatly enhanced available speeds and gate densities in modern digital IC's. Densities reaching 10 million usable gates (with six metal layers) and clock speeds in excess of 1 GHz are available. These capabilities suggest that a programmable imaging architecture for generating false target signatures can be realized with custom ASICs.

A pipelined, all-digital image synthesizer capable of generating false-target images from a series of intercepted ISAR chirp pulses providing a novel radio frequency (RF) imaging decoy capability has been presented and proof of concept completed [4]. This design synthesizes the image of the false target within 512 range bins. Each range bin utilizes two 16-bit binary adders. The design of which is critical to the proper operation of the range bin. The requirement is that these identical adders must be capable of adding two 16-bit binary numbers, providing a 16-bit sum, carry output bit, and overflow indicator bit within one clock cycle with the system being clocked at 600 MHz.

B. PRINCIPAL CONTRIBUTIONS

The first part of the design process included an extensive analysis to utilize the fewest gates in designing the simplest adder that can achieve the 600 MHz goal. SPICE net lists were extracted from these schematic designs and simulations conducted to verify logic functionality and propagation speed. As these designs increased in complexity it was necessary to write a logic minimization computer routine, based on the Quine-McCluskey tabular technique [13]. In sequence, the following designs were explored:

1. 16 x 1-Bit Arithmetic Logic Units (ALU) w/Ripple Carry [12]
2. 4 x 4-Bit Medium Scale Integrated (MSI) Adders w/Ripple Carry [12]
3. 4 x 4-Bit Pseudo-NMOS NOR Logic Adders w/Ripple Carry
4. 4 x 4-Bit CMOS NAND Logic Adders w/Ripple Carry
5. 4 x 4-Bit CMOS NAND Logic Adders w/Carry Look-Ahead

It was necessary to adopt the carry look-ahead technique in order to meet the stated goal. This schematic design underwent extensive SPICE simulation to include being pipelined with registers to verify that it would function within the DIS architecture.

The fabrication mask layout of the verified design was constructed using a CMOS 0.18 micron process utilizing deep sub-micron technology with six metal interconnect layers. The mask layout design was verified by ensuring all design rule checks (DRC) and layout versus schematic (LVS) checks were satisfied. The end-state for this thesis is a fabrication mask layout for a 16-bit adder that accomplishes the stated goal.

C. THESIS OUTLINE

Chapter II will discuss the design process that culminated in the final 16-bit Adder design. Chapter III will detail the schematic design and simulation process. Chapter IV will review the implementation process to achieve the design ready for semiconductor fabrication. Last, Chapter V will offer a concluding summary and recommendations for future work.

II. DESIGN

A. TOOLS

1. Computer Aided Design (CAD) Tools

The Tanner Research, Inc. suite of computer aided design (CAD) tools were used, including S-Edit (schematic editor), L-Edit Pro 8.3 (layout editor), T-Spice Pro (SPICE circuit simulator), LVS (Layout Versus Schematic comparison tool), and W-edit (waveform editor) [8]. In addition, LASI 6, a shareware layout tool was used for preliminary layout [9].

2. MATLAB V5.3

The Math Works, Inc. tool, MATLAB V5.3 was utilized to process SPICE data and for logic minimization [10].

B. TASK(S)

1. Stated Goal

The stated goal is to design and implement a 16-bit full adder that can produce a 16-bit sum, carry output, and overflow bit within one clock cycle at 600 MHz. This design will be implemented through the Metal Oxide Semiconductor Integration Service (MOSIS) at Taiwan Semiconductor Manufacturing Corporation (TSMC) using a 0.18 micron feature size process and SCN6M_DEEP design rules [11].

2. Implied Task(s)

The 16-bit adder will be implemented twice in each of the 512 range bins, for a total of 1024 adders. In general, it can be assumed that the simplest design, utilizing the fewest number of transistors that can accomplish the stated task, is preferable. This type of design would consume the lowest amount of power, smallest amount of area on the chip, minimize the load on the system, and be most readily adaptable to other implementations. For example, a field programmable gate array (FPGA), vice ASIC implementation.

In the stated task, the clock frequency is noted as being 600 MHz. This would correspond to a clock period of:

$$\tau_{CL} = \frac{1}{f} = \frac{1}{600MHz} = 1.67ns \qquad (1)$$

However, in a pipelined very large scale integrated (VLSI) circuit, the input data to the adder will not be available at the start of a clock cycle. Not only will the input pulse shapes be distorted due to transmission timing delays and the capacitive and resistive properties of the physical circuit, but we must account for register propagation and setup times. Recall that in a pipelined design, the adder will be placed between two register arrays. Therefore, the actual available propagation time of the adder will more resemble:

$$Adder\ propagation\ time(t_{pAdd}) \le \tau_{CL} - t_{p\mathrm{Reg}} - t_{s\mathrm{Reg}} \quad (2)$$

It was necessary to simulate these effects in order to ensure that any adder design would meet the requirements of the clock frequency while under load. The distortion of the input was simulated by ramping the input transitions by 100 ps. It was determined that the register setup time (t_{sReg}) was negligible and could be ignored. However, the propagation time of the register was significant and was simulated by utilizing a pulse-shaping circuit consisting of 4 invertors.

A T-Spice simulation of the pulse shaping circuit yielded a register propagation time of approximately 200 ps (Figure 1). Therefore, the propagation time of the adder must be less than or equal to 1.47 ns:

$$t_{pAdd} \le \tau_{CL} - t_{p\mathrm{Reg}} - t_{s\mathrm{Reg}} = 1.67 - 0.2 - 0.0 = 1.47ns \quad (3)$$

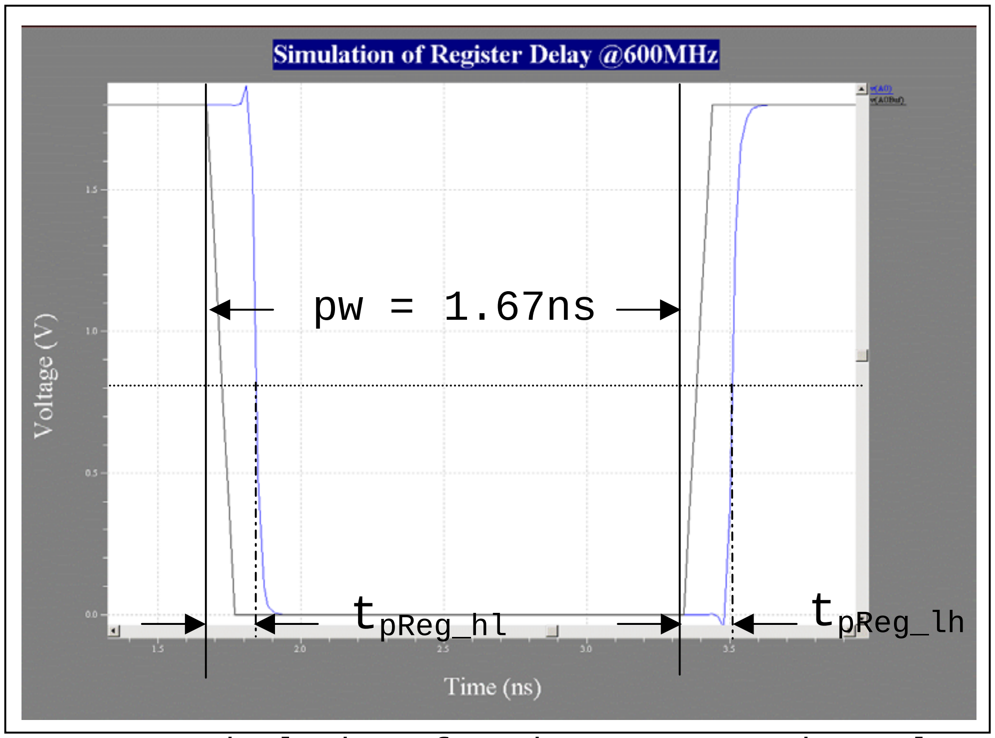


Figure 1. Simulation of Register Propagation Delay.

C. CURRENT SOLUTION

The proof of concept range bin implementation utilized a 16-bit ripple carry adder [7]. Each 1-bit cell produced a 1-bit sum and a 1-bit carry with 3 logic gate delays (3 t_d). Therefore, when the 1-bit cells were connected together to form a 16-bit adder, the delay was 48 t_d (3 x 16) for the 16-bit sum and carry output. Also, there was an additional cost of 3 t_d to realize the overflow bit for a total of 51 t_d. The propagation time for this design is greatly in excess of the design specification. This design could be pipelined into eight stages. However, this will not meet the stated task of performing the addition in one stage.

D. EVOLUTION OF DESIGN

1. 74X283 Medium Scale Integrated (MSI) Adder

The 74X283 is a 4-bit binary adder design that is commercially available (Figure 2). It utilizes the carry look-ahead technique to calculate the 4-bit sum and carry output [12].

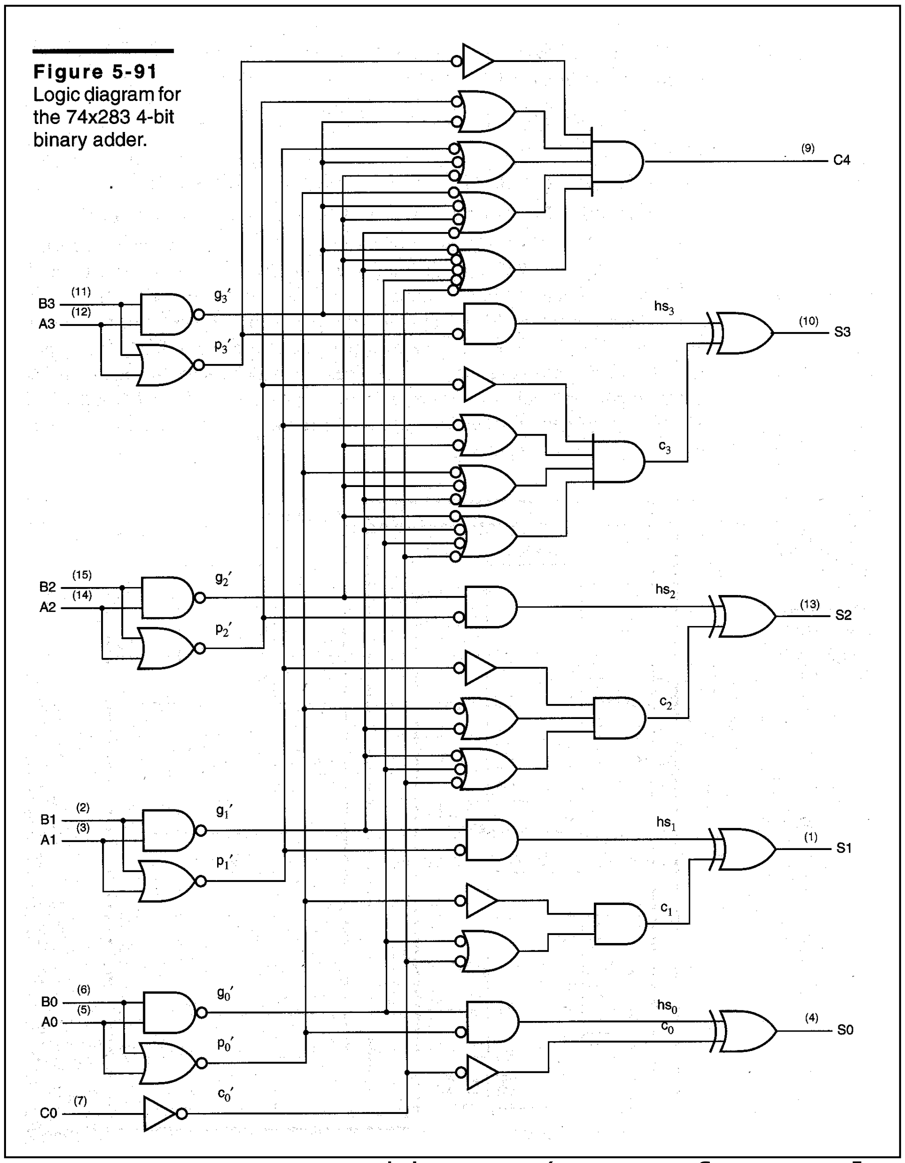


Figure 2. 74X283 MSI 4-bit ALU (From Reference [12]).

An implementation of this design with NOR and NAND gates would have a SUM output delay of 6 t_d and C_4 delay of 3 t_d. A 16-bit group ripple adder can be made from these 4-bit ALU's (Figure 3). The 16-bit SUM delay would be equal to 15 t_d.

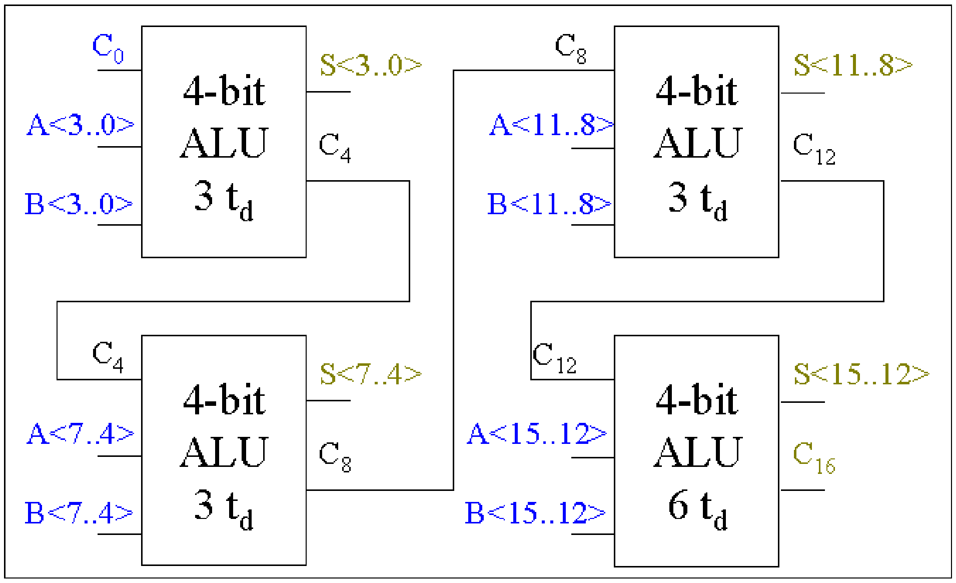


Figure 3. 16-Bit 74X283 w/Ripple Carry Binary Adder.

2. Optimized NOR Gate Logic Design

This MSI design can be leveraged to make it a faster adder. Recall from Figure 2 the logic for the computation of the S_0 bit (Figure 4):

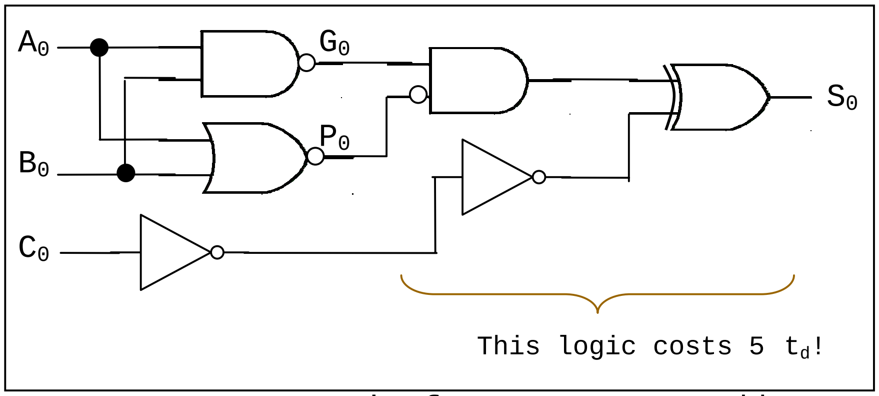


Figure 4. S_0 Logic for 74X283 MSI Adder.

A MATLAB computer program was written (Appendix D) to use the Quine-McCluskey tabular technique [13] to optimize the 4-bit MSI adder with NOR gate logic. Optimized to NOR gate logic, the S_0 bit can be realized by:

$$S_0 = \overline{\overline{(C_0 + G_0 + P_0)} + \overline{(C_0 + \overline{G}_0 + \overline{P}_0)} + \overline{(\overline{C}_0 + \overline{G}_0 + P_0)}} \tag{3}$$

This logic can be implemented as shown in Figure 5 and will save 2 t_d of delay from the 74X283 MSI Adder.

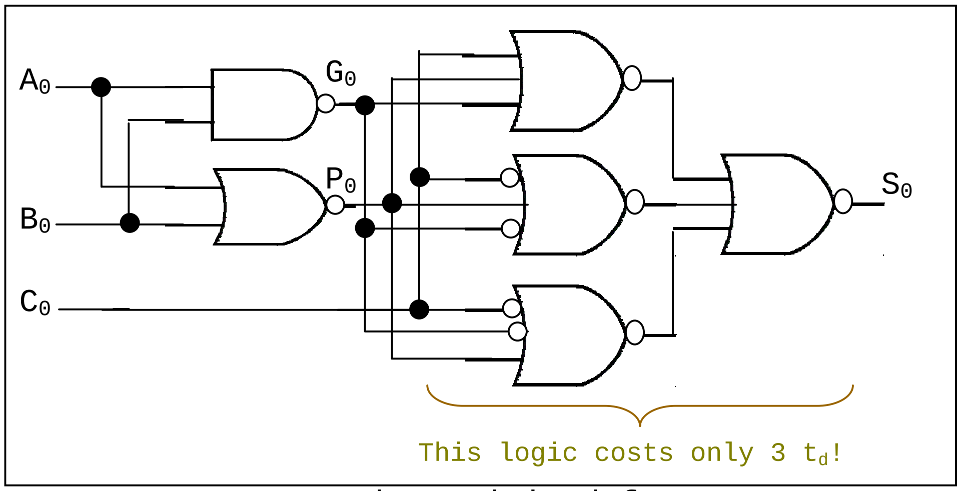


Figure 5. S_0 Logic Optimized for NOR Gates.

The sum and carry output logic equations derived for the 4-bit ALU are shown in Figure 6.

$$S_0 = \overline{\overline{(C_0 + G_0 + P_0)} + \overline{(C_0 + \overline{G}_0 + \overline{P}_0)} + \overline{(\overline{C}_0 + \overline{G}_0 + P_0)}}$$

$$S_1 = \overline{\overline{(C_0 + G_1 + \overline{G}_0 + P_1)} + \overline{(\overline{G}_1 + G_0 + P_1 + P_0)} + \overline{(C_0 + \overline{G}_1 + \overline{G}_0 + \overline{P}_1)} + \overline{(G_1 + \overline{G}_0 + P_1 + \overline{P}_0)} + \overline{(\overline{C}_0 + \overline{G}_1 + P_1 + P_0)} + \overline{(\overline{G}_1 + \overline{G}_0 + \overline{P}_1 + \overline{P}_0)}}$$

$$S_2 = \begin{aligned} &\overline{\overline{(G_2 + \overline{G}_1 + P_2 + \overline{P}_1 + P_0)} + \overline{(C_0 + G_2 + \overline{G}_1 + \overline{G}_0 + P_2)} + \overline{(\overline{G}_2 + G_1 + \overline{G}_0 + P_2 + P_1)} + \overline{(\overline{G}_2 + G_0 + P_2 + P_1 + P_0)} + \overline{(\overline{G}_2 + \overline{G}_1 + \overline{P}_2 + \overline{P}_1 + P_0)}} \\ &\overline{+ \overline{(C_0 + \overline{G}_2 + \overline{G}_1 + \overline{G}_0 + \overline{P}_2)} + \overline{(G_2 + \overline{G}_1 + \overline{G}_0 + P_2 + \overline{P}_0)} + \overline{(\overline{C}_0 + \overline{G}_2 + P_2 + P_1 + P_0)} + \overline{(\overline{G}_2 + \overline{G}_1 + \overline{G}_0 + \overline{P}_2 + \overline{P}_0)}} \end{aligned}$$

$$S_3 = \begin{aligned} &\overline{\overline{(G_3 + \overline{G}_2 + P_3 + \overline{P}_2 + P_1 + P_0)} + \overline{(G_3 + \overline{G}_2 + \overline{G}_0 + P_3 + \overline{P}_2 + P_1)} + \overline{(G_3 + \overline{G}_2 + \overline{G}_1 + P_3 + \overline{P}_1 + P_0)} + \overline{(C_0 + G_3 + \overline{G}_2 + \overline{G}_1 + \overline{G}_0 + P_3)}} \\ &\overline{+ \overline{(\overline{G}_3 + G_2 + \overline{G}_1 + P_3 + P_2 + P_0)} + \overline{(\overline{G}_3 + G_2 + \overline{G}_1 + \overline{G}_0 + P_3 + P_2)} + \overline{(\overline{G}_3 + \overline{G}_2 + \overline{P}_3 + \overline{P}_2 + P_1 + P_0)} + \overline{(\overline{G}_3 + G_1 + \overline{G}_0 + P_3 + P_2 + P_1)}} \\ &\overline{+ \overline{(\overline{G}_3 + \overline{G}_2 + \overline{G}_0 + \overline{P}_3 + \overline{P}_2 + P_1)} + \overline{(\overline{G}_3 + G_0 + P_3 + P_2 + P_1 + P_0)} + \overline{(\overline{G}_3 + \overline{G}_2 + \overline{G}_1 + \overline{P}_3 + \overline{P}_1 + P_0)} + \overline{(C_0 + \overline{G}_3 + \overline{G}_2 + \overline{G}_1 + \overline{G}_0 + \overline{P}_3)}} \\ &\overline{+ \overline{(G_3 + \overline{G}_2 + \overline{G}_1 + \overline{G}_0 + P_3 + \overline{P}_0)} + \overline{(\overline{C}_0 + \overline{G}_3 + P_3 + P_2 + P_1 + P_0)} + \overline{(\overline{G}_3 + \overline{G}_2 + \overline{G}_1 + \overline{G}_0 + \overline{P}_3 + \overline{P}_0)}} \end{aligned}$$

$$C_4 = \begin{aligned} &\overline{\overline{(\overline{G}_3 + \overline{P}_3 + P_2 + P_1 + P_0)} + \overline{(\overline{G}_3 + \overline{G}_0 + \overline{P}_3 + P_2 + P_1)} + \overline{(\overline{G}_3 + \overline{G}_1 + \overline{P}_3 + P_2 + P_0)} + \overline{(\overline{G}_3 + \overline{G}_1 + \overline{G}_0 + \overline{P}_3 + P_2)} + \overline{(\overline{G}_3 + \overline{G}_2 + \overline{P}_2 + P_1 + P_0)}} \\ &\overline{+ \overline{(\overline{G}_3 + \overline{G}_2 + \overline{G}_0 + \overline{P}_2 + P_1)} + \overline{(\overline{G}_3 + \overline{G}_2 + \overline{G}_1 + \overline{P}_1 + P_0)} + \overline{(C_0 + \overline{G}_3 + \overline{G}_2 + \overline{G}_1 + \overline{G}_0)} + \overline{(\overline{G}_3 + \overline{G}_2 + \overline{G}_1 + \overline{G}_0 + \overline{P}_0)}} \end{aligned}$$

Figure 6. 4-Bit ALU Optimized With NOR Logic.

A 4-bit NOR Logic ALU has a delay of 4 t_d for sum and carry outputs. However, if they were utilized similar to Figure 3, the upper level ALU's will only have a delay of 3 t_d due to the propagate and generate inputs to the NOR logic already being produced. Therefore, as per Figure 7, a 16-bit NOR Gate Adder would only have a delay of 13 t_d.

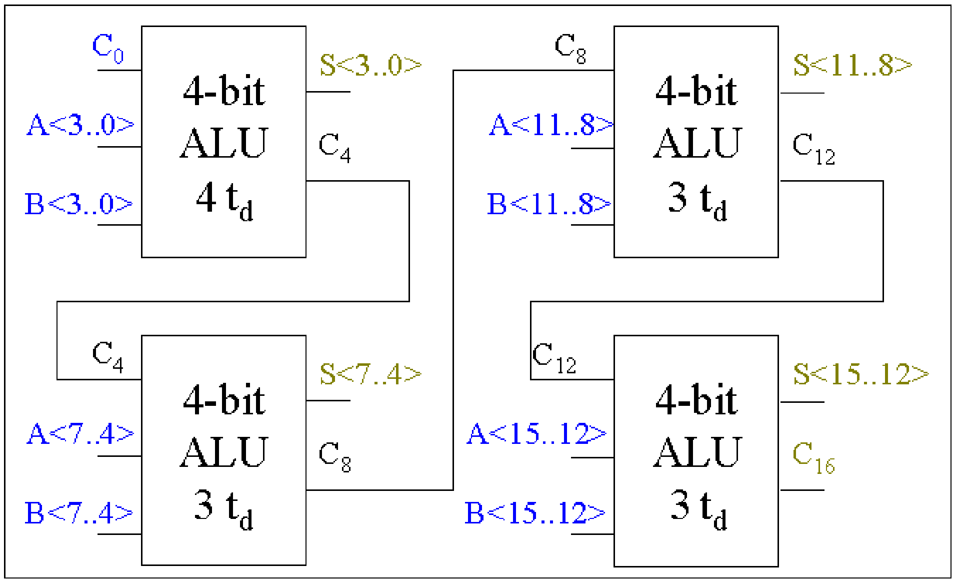


Figure 7. 16-Bit NOR Logic Ripple Carry Adder.

The LASI CAD tool was utilized to produce a semiconductor fabrication mask layout for the 4-Bit NOR Logic ALU utilizing pseudo-NMOS NOR gates (Figure 8).

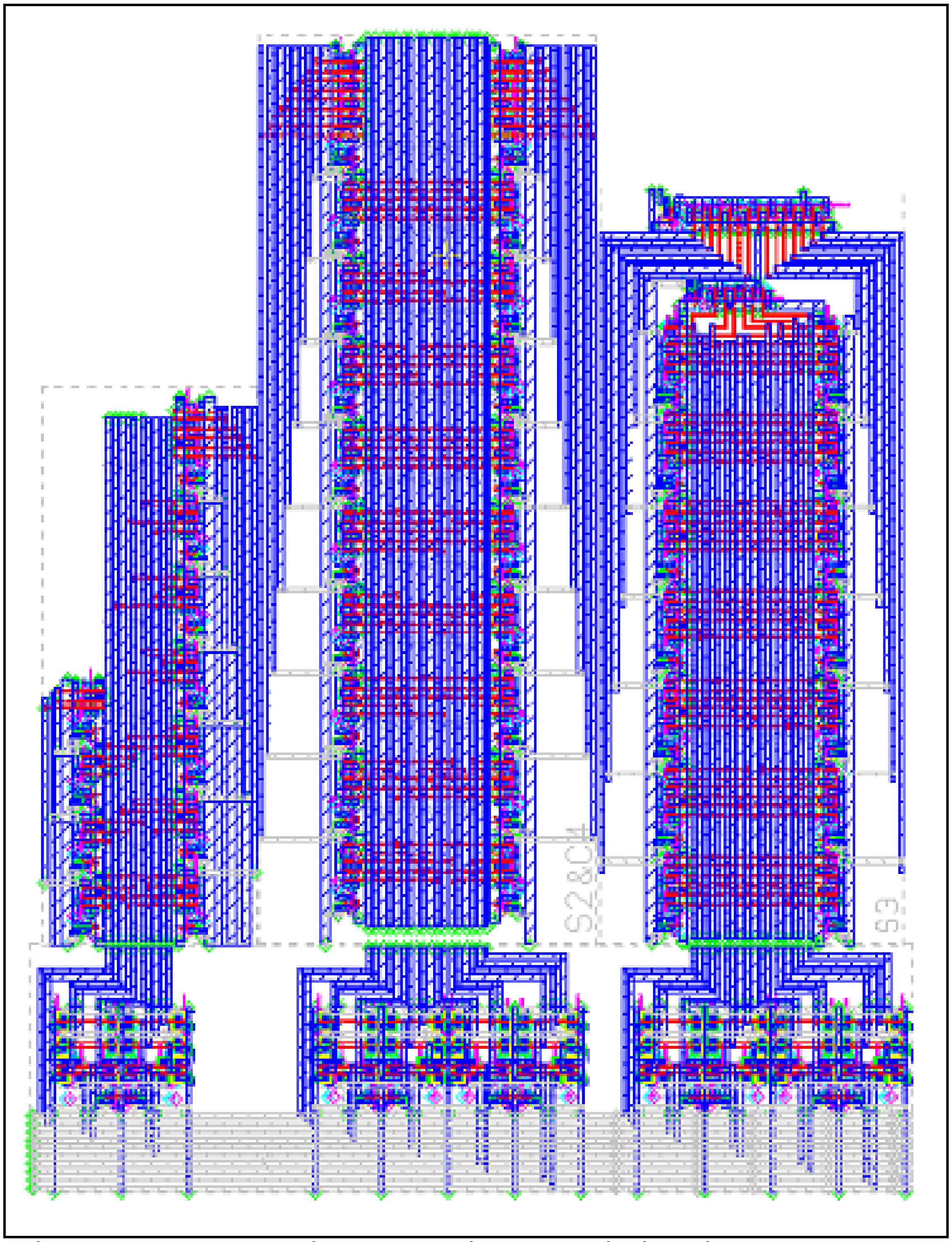


Figure 8.   4-Bit NOR Logic ALU Fabrication Layout.

The Spice net list of the psuedo-NMOS NOR layout (Appendix E) was tested to determine if the 16-bit NOR Logic Ripple Carry Adder would be an effective design. It was found that it easily exceeded the stated goal, even meeting a 1 GHz clock speed. In regards to the implied tasks, it used only 2316 transistors and could be implemented very compactly with the Tanner Tools. However, it consumed a significant amount of power. A comparison of an application of 1024 16-bit NOR Adders with the Pentium III processor is presented in Table 1 [14].

Item	Feature Size (um)	Qty FET (M)	Speed (MHz)	Vdd (V)	Avg Power (W)
1024 NOR Adders	0.18	2.4	600	1.8	12
Pentium III	0.18	10	700	1.6	15

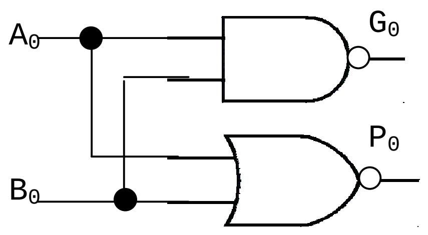
Table 1. 1024 16-Bit NOR Adders vs Pentium III Power Consumption Comparison.

The Pentium III requires active cooling; i.e. a fan. It is not known at this writing if the final design of the radar imaging chip will require active cooling. However, this level of power consumption for just the 16-bit adder components (approximately a third of the total number of transistors used in the radar ASIC) is not acceptable. Unfortunately, when ‘regular’ CMOS NOR gates are substituted in this design, it doesn’t meet the clock rate minimum specification.

3. Optimized NAND Gate Logic Design

In general, designs using NAND gates are faster than the NOR gate equivalent. Therefore, the Quine-McCluskey logic minimization program was used to determine equivalent NAND gate logic for the 4-bit adder. The resultant logic equations were not practical. They were too big, using too many variables, causing a number of gate fanout problems.

However, in the calculation of the input generate and propagate variables, there was a "don't care" result (Figure 9).



A_0	B_0	G_0	P_0
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0

Never have

G_0	P_0
0	1

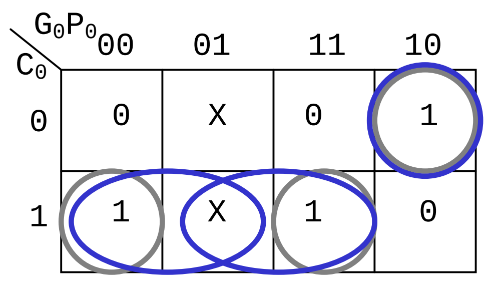
Figure 9. Don't Care Result in G and P Calculation.

Examining the S_0 calculation, it can be seen how to leverage this "don't care" in order to minimize the logic equation (Figure 10).

Expansion of truth table to include C_0 & S_0

C_0	A_0	B_0	G_0	P_0	S_0
0	0	0	1	1	0
0	0	1	1	0	1
0	1	0	1	0	1
0	1	1	0	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	0	0	1

…and now the Karnough map for S_0



C_0 \ G_0P_0	00	01	11	10
0	0	X	0	1
1	1	X	1	0

Minterms: $S_0 = C_0\overline{G_0}\overline{P_0} + C_0G_0P_0 + \overline{C_0}G_0\overline{P_0}$

Using don't cares: $S_0 = C_0\overline{G_0} + C_0P_0 + \overline{C_0}G_0\overline{P_0}$

Figure 10. Use of Don't Care Values in Sum Terms.

Expanding this technique to all sum terms for NAND gate logic yields the logic equations in Figure 11.

$$S_0 = \overline{((\overline{C_0\overline{G_0}})(\overline{C_0P_0})(\overline{\overline{C_0}G_0\overline{P_0}}))}$$

$$S_1 = \overline{((\overline{\overline{G_1}\,\overline{G_0}})(\overline{\overline{G_0}P_1})(\overline{\overline{C_0}G_1G_0\overline{P_1}})(\overline{C_0\overline{G_1}\,\overline{P_0}})(\overline{G_1\overline{P_1}P_0})(\overline{C_0P_1\overline{P_0}}))}$$

$$S_2 = \overline{((\overline{\overline{G_2}\,\overline{G_1}})(\overline{\overline{G_2}\,\overline{G_0}\,\overline{P_1}})(\overline{\overline{G_1}P_2})(\overline{G_2\overline{P_2}P_1})(\overline{\overline{G_0}P_2\overline{P_1}})(\overline{\overline{C_0}G_2G_1G_0\overline{P_2}})(\overline{C_0\overline{G_2}\,\overline{P_1}\,\overline{P_0}})(\overline{G_2G_1\overline{P_2}P_0})(\overline{C_0P_2\overline{P_1}\,\overline{P_0}}))}$$

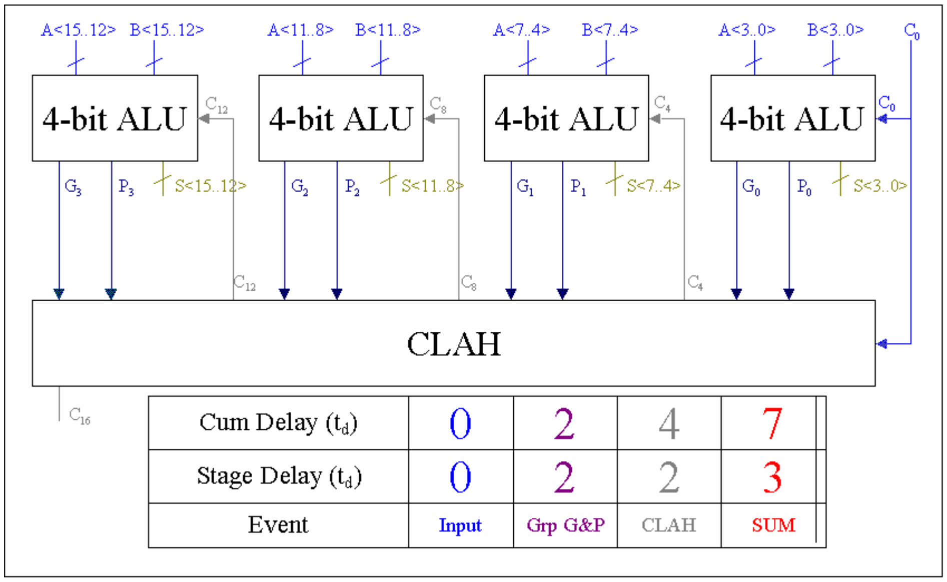
$$S_3 = \left(\begin{array}{c}\overline{(\overline{\overline{G_3}\,\overline{G_2}G_1})(\overline{\overline{G_3}\,\overline{G_1}G_0\overline{P_2}})(\overline{\overline{G_3}\,\overline{G_0}\,\overline{P_2}\,\overline{P_1}})(\overline{\overline{G_2}P_3\overline{P_0}})(\overline{\overline{G_2}G_1P_3G_0})(\overline{G_3\overline{P_3}P_2\overline{P_1}})(\overline{\overline{G_1}P_3\overline{P_2}})(\overline{G_3G_2P_1\overline{P_3}})} \\ (\overline{G_3\overline{G_0}P_3\overline{P_2}\,\overline{P_1}})(\overline{\overline{C_0}G_3G_2G_1G_0\overline{P_3}})(\overline{C_0\overline{G_3}\,\overline{P_2}\,\overline{P_1}\,\overline{P_0}})(\overline{G_3G_2G_1\overline{P_3}P_0})(\overline{C_0P_3\overline{P_2}\,\overline{P_1}\,\overline{P_0}})\end{array}\right)$$

Figure 11. 4-Bit ALU Optimized With NAND Logic.

Although this technique garnered considerable savings in transistors and solved the fanout problem, simulations of a 16-bit adder built using these 4-bit ALU's with ripple carry between the 4-bit groups were still not fast enough. Therefore, it was necessary to use Carry Look-Ahead (CLAH).

4. Carry Look-Ahead Unit

The delay path for a 16-bit adder using the carry look-ahead technique is summarized in Figure 12 [12]. It has a delay of 7 t_d for sum and carry output.



Cum Delay (t_d)	0	2	4	7
Stage Delay (t_d)	0	2	2	3
Event	Input	Grp G&P	CLAH	SUM

Figure 12. Delay Path for 16-Bit Adder w/CLAH.

The carry look-ahead technique introduces two new circuits to the adder design. A Group Generate (Grp_G) and Group Propagate (Grp_P) circuit is added to the 4-bit ALU in addition to a CLAH for the 16-bit Adder. The Quine-McCluskey minimization program was utilized to produce the logic terms for both circuits (Figures 13 and 14).

$$C_4 = (\overline{\overline{G_0}(\overline{C_0 P_0})})$$

$$C_8 = (\overline{\overline{G_1}(\overline{G_0 P_1})(\overline{C_0 P_0 P_1})})$$

$$C_{12} = (\overline{\overline{G_2}(\overline{G_1 P_2})(\overline{G_0 P_1 P_2})(\overline{C_0 P_0 P_1 P_2})})$$

$$C_{16} = (\overline{\overline{G_3}(\overline{G_2 P_3})(\overline{G_1 P_2 P_3})(\overline{G_0 P_1 P_2 P_3})(\overline{C_0 P_0 P_1 P_2 P_3})})$$

Figure 13. CLAH NAND Gate Logic Terms.

$$Grp\_G = \left(\overline{\begin{array}{l}(\overline{A_0 B_0 B_1 B_2 B_3})(\overline{A_1 B_1 B_2 B_3})(\overline{A_0 A_2 B_0 B_2 B_3})(\overline{A_2 B_2 B_3})(\overline{A_0 A_2 B_0 B_1 B_3})(\overline{A_1 A_2 B_1 B_3})(\overline{A_0 A_1 A_2 B_0 B_3}) \\ (\overline{A_3 B_3})(\overline{A_0 A_3 B_0 B_1 B_2})(\overline{A_1 A_3 B_1 B_2})(\overline{A_0 A_2 A_3 B_0 B_2})(\overline{A_2 A_3 B_2})(\overline{A_0 A_2 A_3 B_0 B_1})(\overline{A_1 A_2 A_3 B_1})(\overline{A_0 A_1 A_2 A_3 B_0})\end{array}}\right)$$

$$Grp\_P = (\overline{(\overline{A_0 + B_0}) + (\overline{A_1 + B_1}) + (\overline{A_2 + B_2}) + (\overline{A_3 + B_3})})$$

Figure 14. Group G and P Logic Terms.

Simulations of this design indicated that it would meet the stated goal of completing the 16-bit sum and carry output in one clock cycle at 600 MHz. However, there remained the necessity to design the overflow detector. An overflow condition exists if:

$$Overflow(Ov) = C_{15} \oplus C_{16} \quad (4)$$

In order to facilitate the future reuse of the 4-bit ALU design, the decision was made to incorporate the overflow detection circuit as a component of the 4-bit ALU. Therefore, in this particular implementation of the 16-bit adder, only the overflow output of the uppermost 4-bit ALU would be used.

The Quine-McCluskey minimization program was used to determine the logic terms. The results were not encouraging. There were too many product terms to meet fanout restrictions. In an attempt to reduce the number of variables from nine to eight and simplify the complexity of the minimization, two separate minimization calculations were performed. The C_0 bit was set high (1.8 volts) for the first calculation and low (0.0 volts) for the second. Each calculation resulted in eight 4-input product terms. Six product terms were identical for both calculations.

It was decided to leverage these shared product terms and construct an overflow circuit that simultaneously calculates two potential answers; C_0is1 and C_0is0 (Figure 15). When the C_0 becomes available, it is used as a control signal to multiplex the correct answer signal to the overflow output. With the use of 2X1 multiplexer using active-pass gates, this circuit produces the overflow output with no additional delay than the sum and carry output.

$$C_0is0 = \overline{((\overline{\overline{G_3}G_2G_1G_0})(\overline{\overline{G_0}P_3\overline{P_2}\,\overline{P_1}})(\overline{\overline{G_3}P_2\overline{P_1}\,\overline{P_0}})(\overline{\overline{G_3}G_0P_2\overline{P_1}})(\overline{\overline{G_3}G_2P_1\overline{P_0}})(\overline{\overline{G_2}G_1P_3\overline{P_0}})(\overline{\overline{G_2}G_1G_0P_3})(\overline{\overline{G_1}G_0P_3\overline{P_2}}))}$$

$$C_0is1 = \overline{((\overline{\overline{G_3}G_2G_1P_0})(\overline{P_3\overline{P_2}\,\overline{P_1}\,\overline{P_0}})(\overline{\overline{G_3}P_2\overline{P_1}\,\overline{P_0}})(\overline{\overline{G_3}G_0P_2\overline{P_1}})(\overline{\overline{G_3}G_2P_1\overline{P_0}})(\overline{\overline{G_2}G_1P_3\overline{P_0}})(\overline{\overline{G_2}G_1G_0P_3})(\overline{\overline{G_1}G_0P_3\overline{P_2}}))}$$

Figure 15. Overflow Circuit 2x1 Multiplexer Signals.

5. Gate Design

Once the adder architecture had been determined, consideration was given to the design of the individual logic gates. The basic building blocks of a CMOS logic gate are field effect transistors (FET); specifically N-channel (NFET) and P-channel (PFET) [15]. A decision was made to optimize the respective sizes of the NFETs and PFETs in each gate, first toward balancing the noise margins of the gates (Figure 16). This is important so that the gate is tolerant of small changes to power supply and input voltage levels, possibly caused by noise.

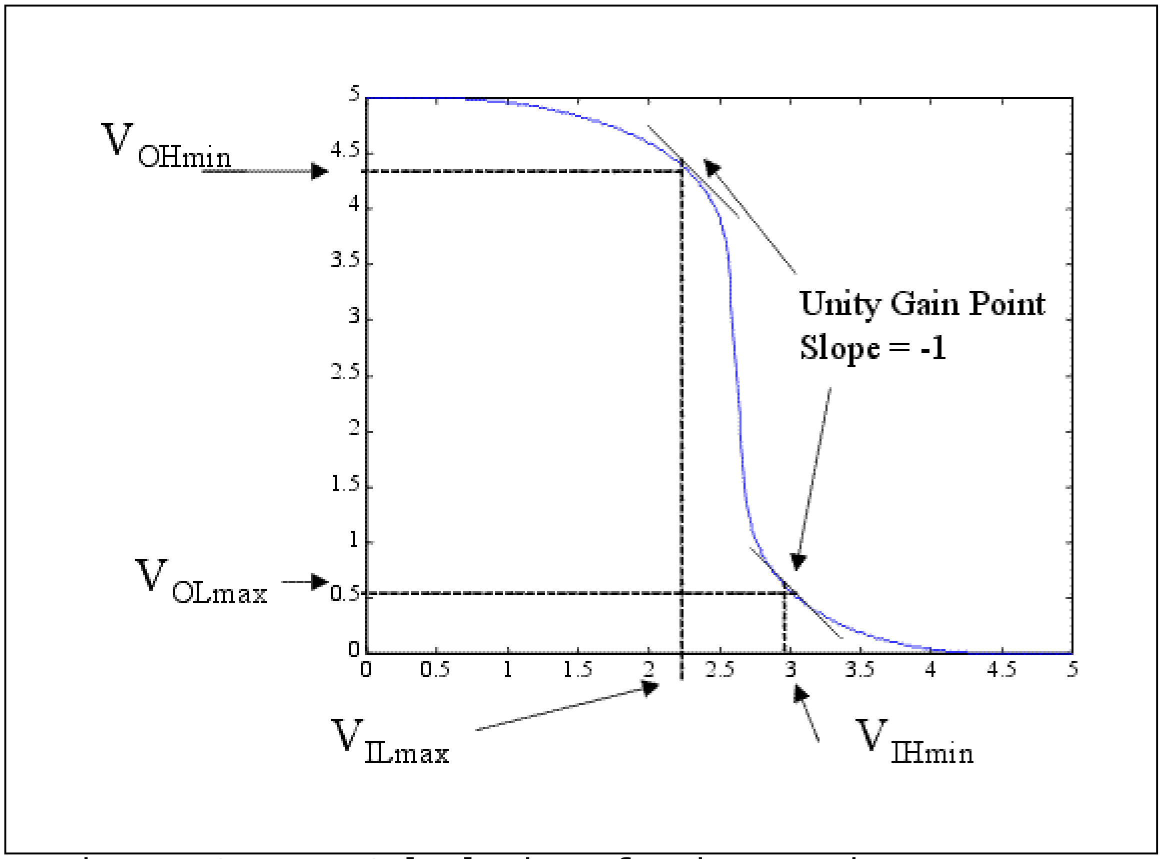


Figure 16. Calculation of Noise Margins.

In a 1.8 volt circuit, the optimum noise margin would be 0.9 volts. For example, a gate with a skewed noise margin low (NM_l) of 0.4 volts would perceive an input to be a logic one if that input voltage inadvertently rose from 0.0 to 0.4 volts. This could cause the gate output to change and result in a logic error. Table 2 lists the gate noise margins for this design.

In only one case is a gate noise margin more than 0.3 volts from optimum. In the case of the 4-input NOR gate, the noise margin low had to be skewed in order that the second criteria, that of gate propagation time, be met.

The AC Transient Analysis data for the gates is included in Table 3.

Gate	# Inputs	NM_l	NM_h	NFET Size (lambda)	PFET Size (lambda)
INV	1	0.8487	0.9808	5	14
NAND	2	0.8487	0.9808	5	10
NAND	3	0.9501	0.8395	5	10
NAND	4	0.8826	0.8634	5	8
NAND	5	0.9808	0.7506	5	8
NAND	6	0.7025	0.9833	5	5
NAND	8	0.8532	0.8028	5	5
NAND	9	0.9272	0.7245	5	5
NAND	13	0.7450	0.8609	10	5
NAND	15	0.9268	0.6764	10	5
NOR	2	0.8228	0.8867	5	17
NOR	4	0.3967	1.1056	5	17
MUX	2+control	0.9876	0.9282	5	14

Table 2. Gate Noise Margins.

Gate	#inputs	t_f	t_r	t_{phl}	t_{plh}
INV	1	0.0110	0.0130	0.0100	0.0110
NAND	2	0.0110	0.0130	0.0100	0.0100
NAND	3	0.0170	0.0150	0.0140	0.0120
NAND	4	0.0240	0.0160	0.0150	0.0130
NAND	5	0.0330	0.0180	0.0180	0.0150
NAND	6	0.0360	0.0210	0.0140	0.0170
NAND	8	0.0560	0.0250	0.0170	0.0200
NAND	9	0.0690	0.0280	0.0180	0.0210
NAND	13	0.1137	0.0400	0.0150	0.0300
NAND	15	0.1540	0.0470	0.0180	0.0330
NOR	2	0.0210	0.0250	0.0200	0.0160
NOR	4	0.0550	0.0820	0.0390	0.0420
MUX	2+control	0.0140	0.0180	0.0100	0.0100

Table 3. Gate AC Transient Data.

In order to balance the noise margins for the 4-input NOR gate, a significant increase in the size of the PFETs was required. However, the AC transient analysis determined that the fall time (t_f) was very sensitive to any increase in PFET size. The speed of this gate is particularly important for the entire system delay because it is used in the Group Generate and Propagate circuit to calculate the Grp_P output that goes to the CLAH. It was decided to allow the slight increase in noise margin skewing to minimize the gate propagation delay.

III. SCHEMATIC DESIGN OF 16-BIT ADDER

A. OVERVIEW

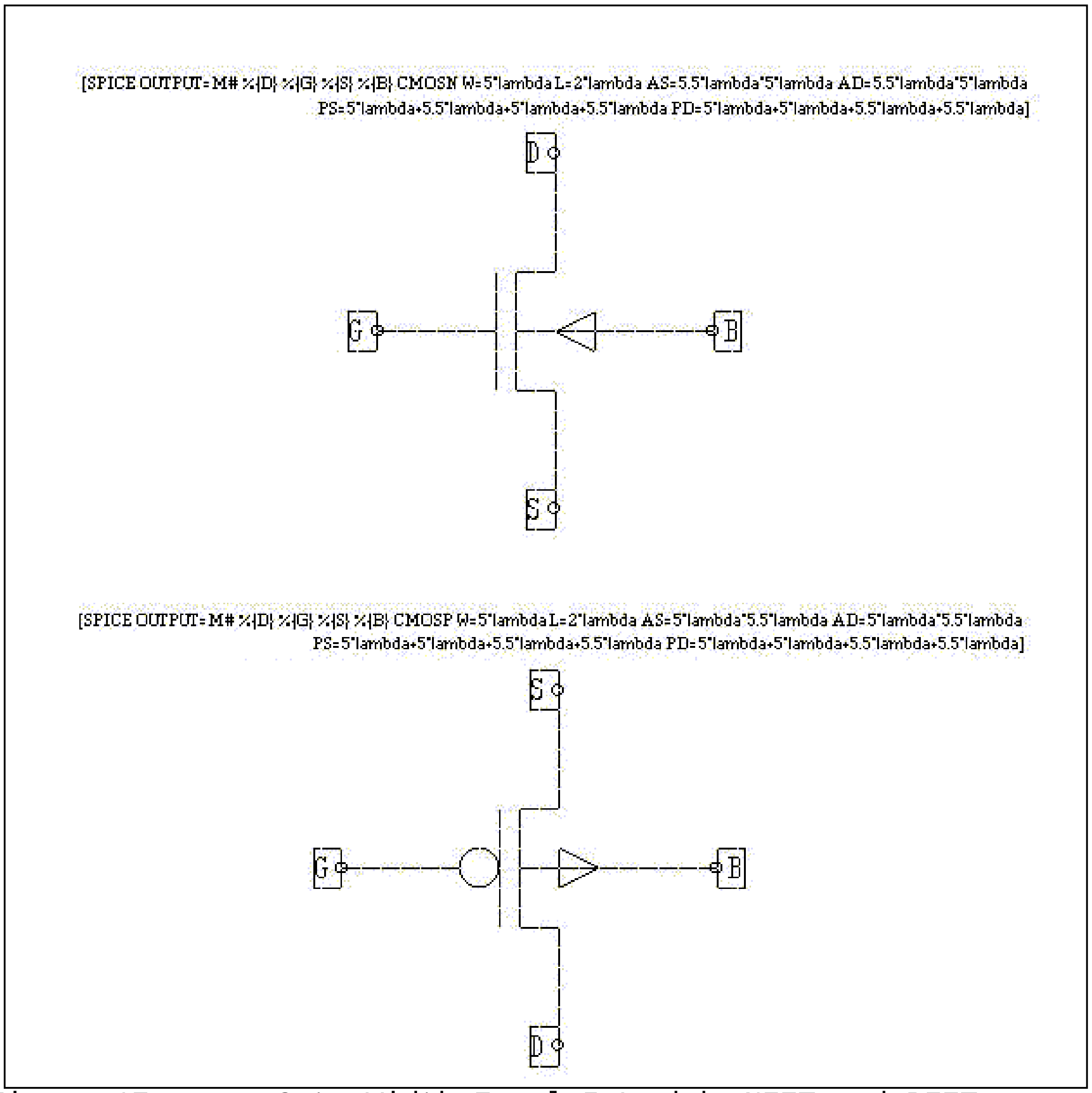
The design process used the Tanner Tools schematic editor (S-Edit) to assemble the sub-circuits and eventually the 16-bit adder. This design process is hierarchical in nature. In sequential fashion, the lower level cells were constructed and integrated into logic gates, which were used in more complex sub-circuits. At each step, a SPICE net list was extracted from S-Edit to ensure the electrical connectivity and logic functionality of each sub-circuit. This process concluded with a SPICE net list being extracted for the 16-Bit Adder and simulated to ensure that it is fault free and meets the stated task of functioning with a 600 MHz clock.

B. HIERARCHICAL SCHEMATIC DESIGN

1. Level One – Transistors

This design utilizes 2 sizes of NFETs and 5 sizes of PFET’s. In all cases the gate length is 2 lambda, the minimum allowed for this fabrication process. NFETs with gate widths of 5 and 10 lambda and PFETs with gate widths of 5, 8, 10, 14, and 17 lambda were utilized.

Figure 17 is an example of an NFET and PFET design. Note that each FET includes specifications for the calculation of source and drain area (AS and AD) and perimeter (PS and PD). Since this design is fully scalable, these parameters are calculated using the scaling parameter lambda. This will provide a more accurate simulation of the timing characteristics of the circuit and prove critical toward determining the realistic speed of the 16-bit Adder. See Appendix A for a complete set of FET schematics.



```
[SPICE OUTPUT= M# %{D} %{G} %{S} %{B} CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
                    PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda]
```

```
[SPICE OUTPUT= M# %{D} %{G} %{S} %{B} CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
                    PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda]
```

Figure 17. Gate Width Equal 5 Lambda NFET and PFET.

2. Level Two – Logic Gates

These transistors were utilized to assemble a number of logic gates. Table 4 specifies the logic gates that were used in this design. Figure 18 is an example of NAND and NOR gate design. See Appendix A for a complete set of logic gate schematics.

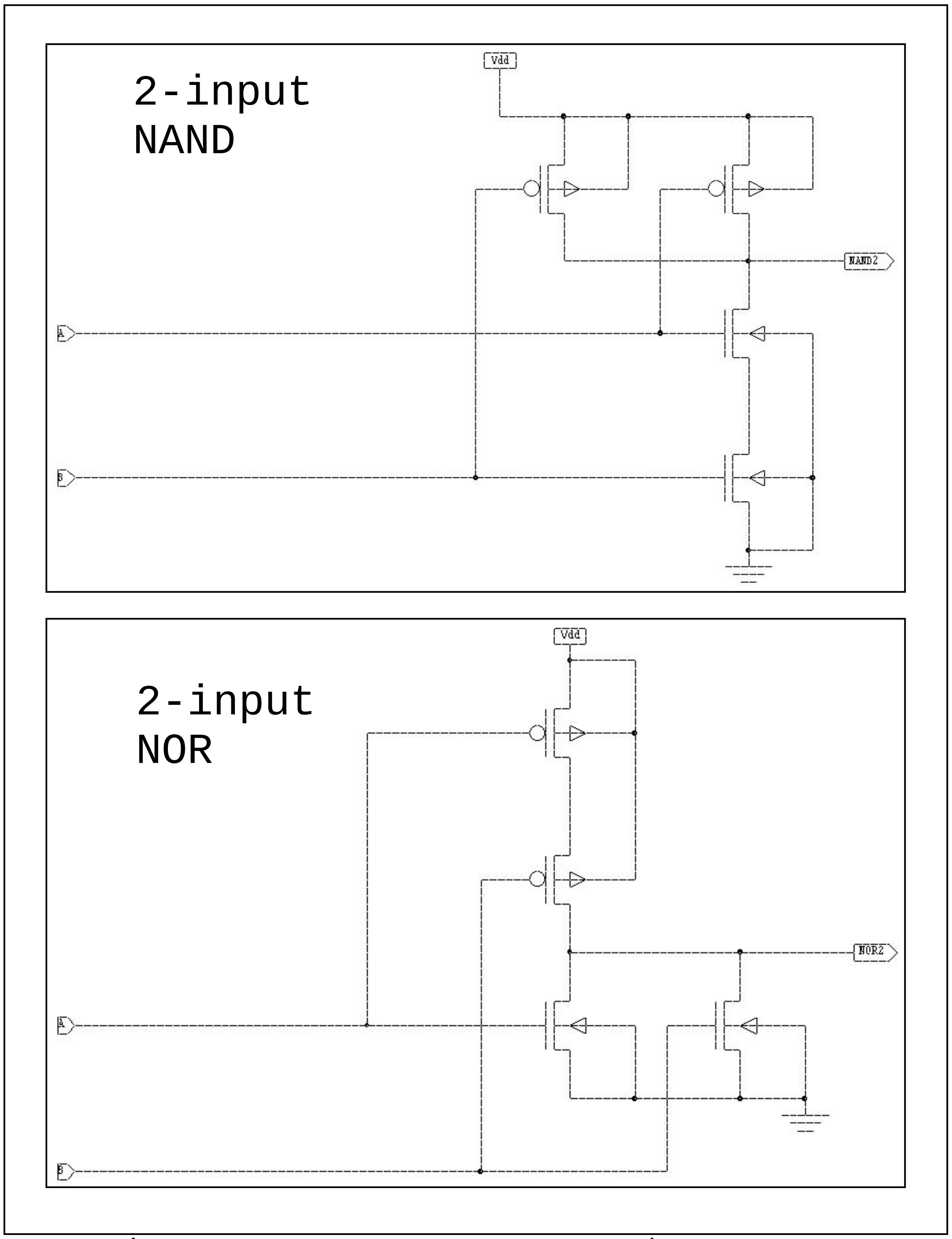


Figure 18. 2-Input NAND and NOR.

Type Gate	Number of Inputs	NFET Size (lambda)	PFET Size (lambda)
INV	2	5	14
NAND	2	5	10
NAND	3	5	10
NAND	4	5	8
NAND	5	5	8
NAND	6	5	8
NAND	8	5	5
NAND	9	5	5
NAND	13	10	5
NAND	15	10	5
NOR	2	5	17
NOR	4	5	17
MUX	2 + control	5	14

Table 4. Logic Gates.

3. Level Three – Secondary Sub-Circuits

a. 4-Bit ALU Group Generate and Propagate (Group GP)

The logic equations for the Group GP were presented in Figure 14 and are realized in Figure 19.

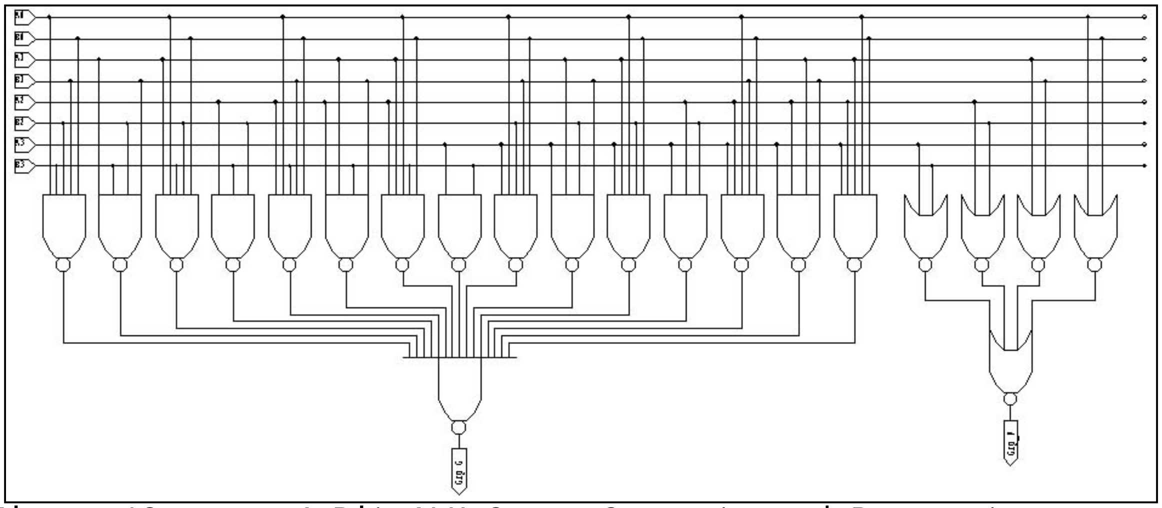


Figure 19. 4-Bit ALU Group Generate and Propagate.

Some experimentation was conducted in realizing these logic functions with combinational gates. One (large) gate would be constructed for the Grp_G output and one for the Grp_P output. The immediate advantage is that there are less than half the number of transistors in the combinational gate than in the logic gates that are used in Figure 19 (Figures 20 and 21).

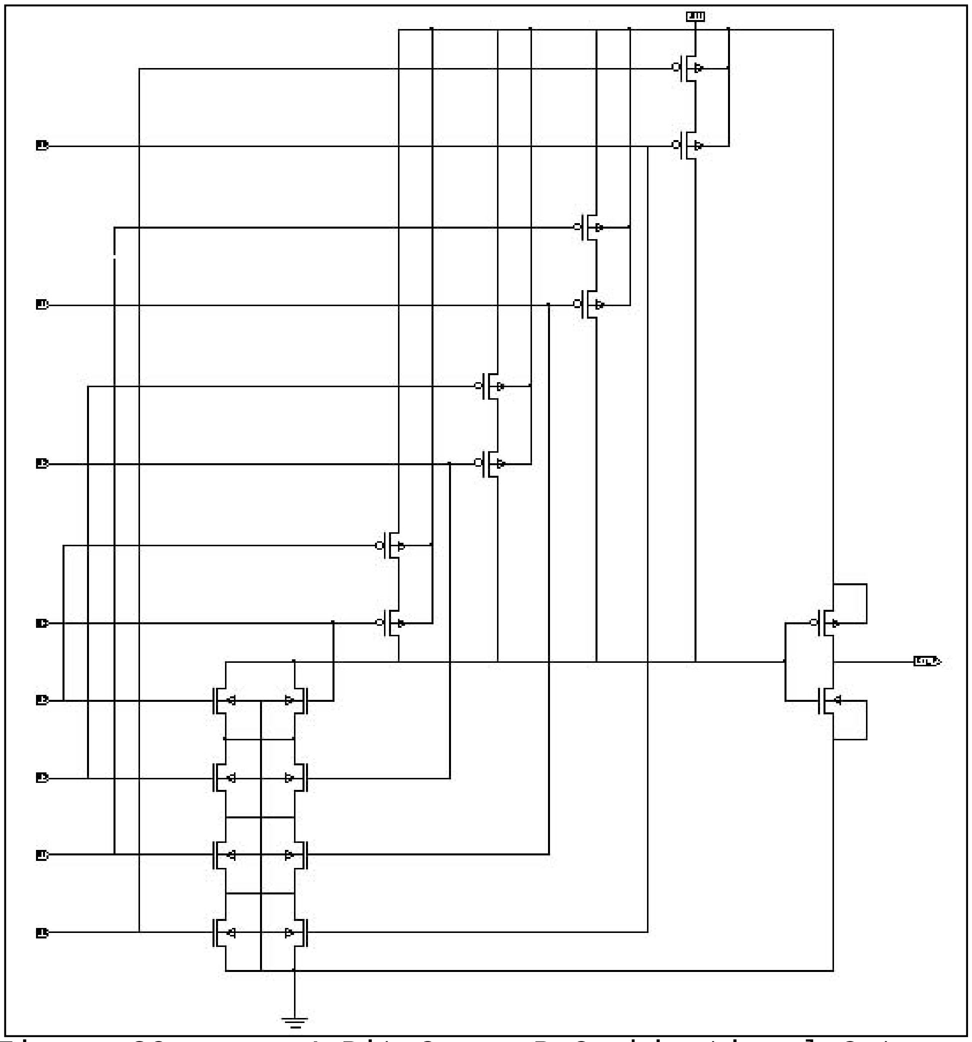


Figure 20. 4-Bit Group P Combinational Gate.

During initial testing, these gates appeared to be a vast improvement in space and speed. Unfortunately, when they were placed under load, they became slower than the logic gate versions.

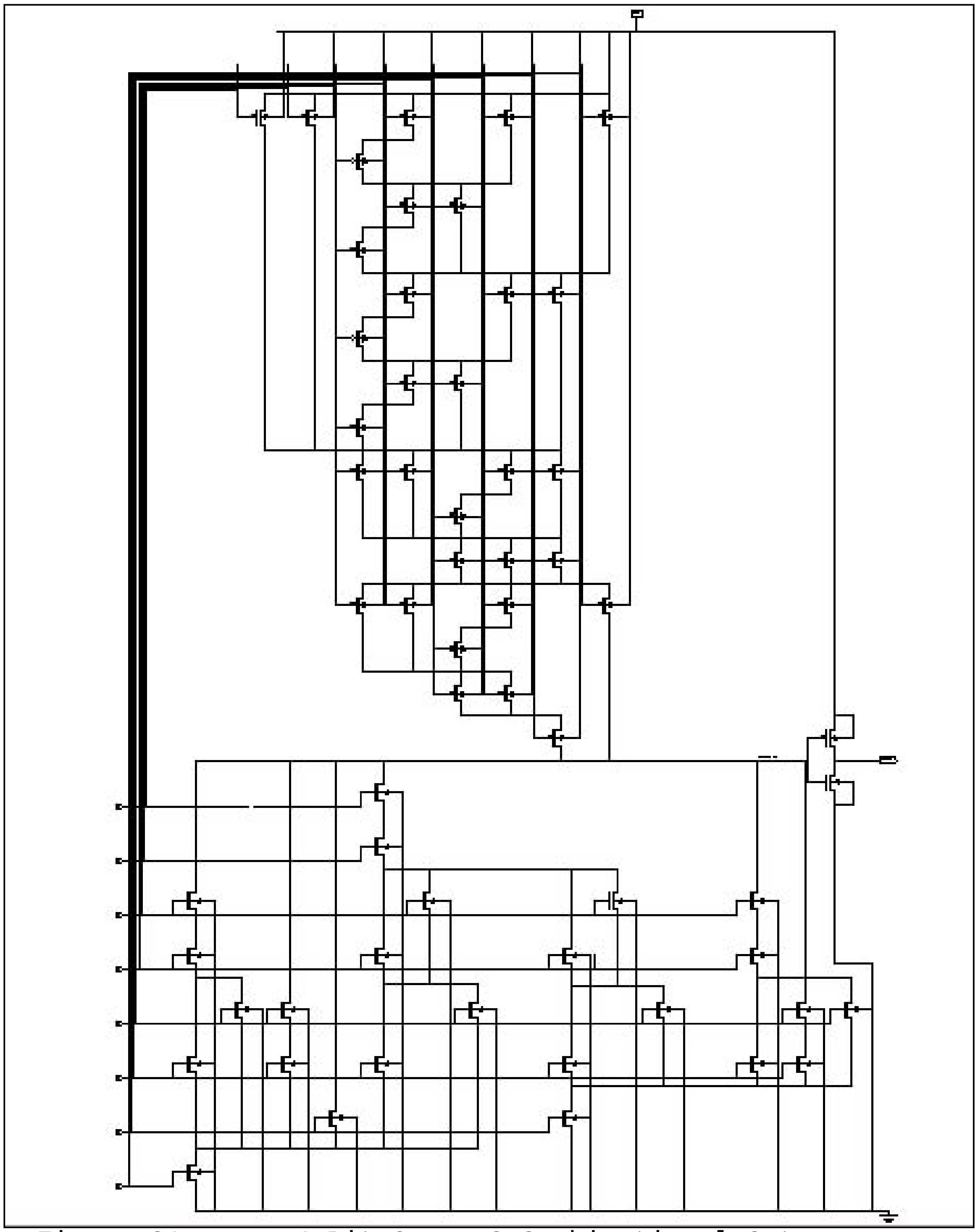


Figure 21. 4-Bit Group G Combinational Gate.

b. 4-Bit ALU Generate and Propagate (ALU GP)

The 1-Bit ALU Generate and Propagate design was demonstrated in Figures 4,5, and 8. A schematic diagram for a 1-Bit ALU GP was assembled and then expanded to the 4-Bit ALU GP (Figure 22).

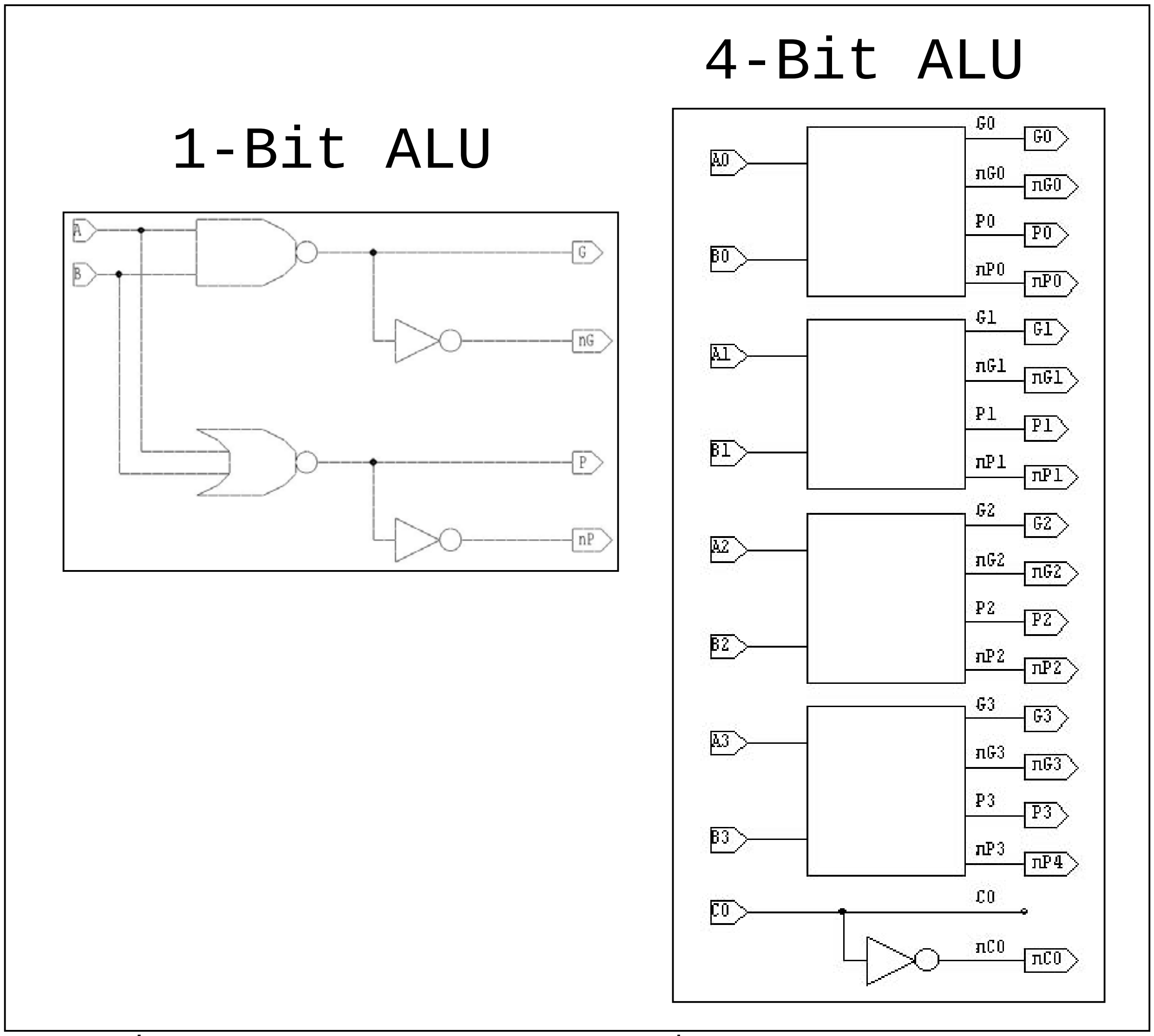


Figure 22. ALU Generate and Propagate.

c. 4-Bit ALU Overflow

The logic equations for this sub-circuit were presented in Figure 15 and are realized in Figure 23. As previously discussed, the two signals Cis0 and Cis1 are multiplexed with C_0 being the control variable.

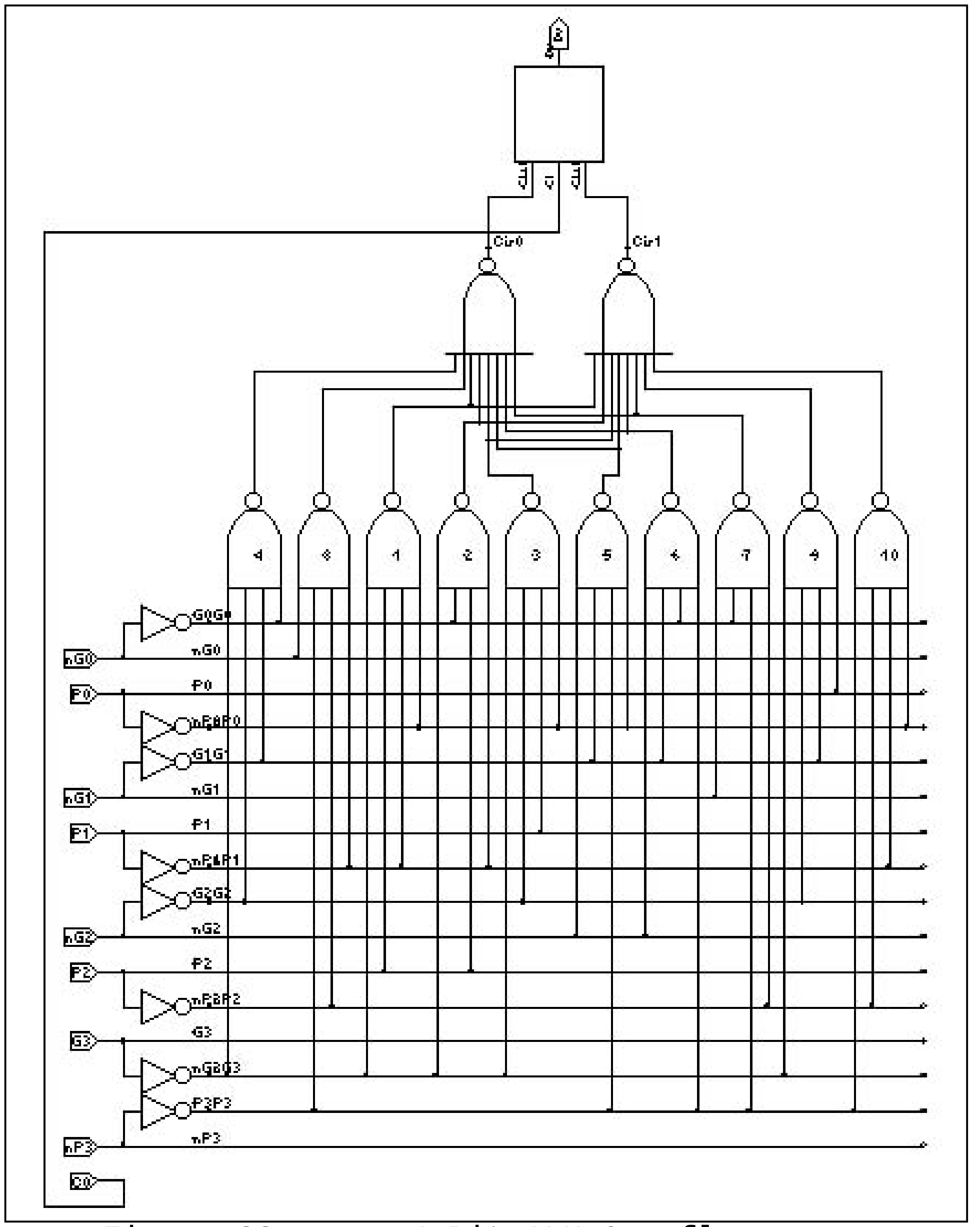


Figure 23. 4-Bit ALU Overflow.

d. 16-Bit Adder Carry Look-Ahead

The logic equations for this sub-circuit were presented in Figure 13 and are realized in Figure 24.

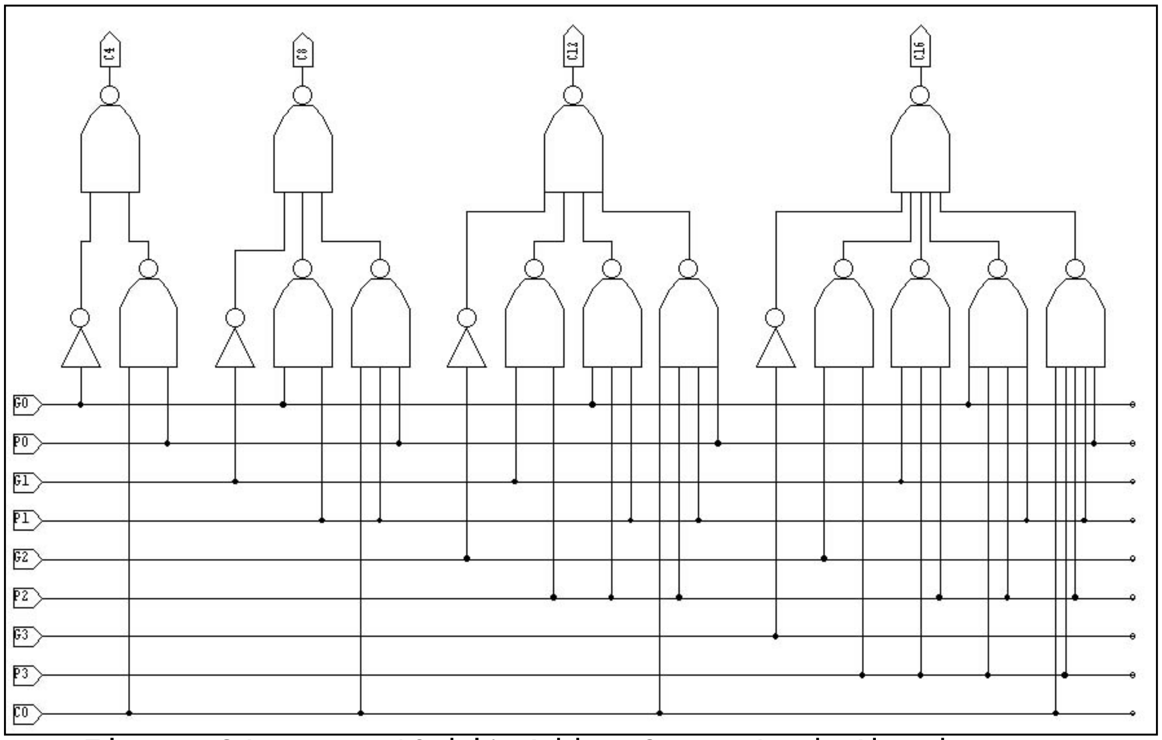


Figure 24. 16-bit Adder Carry Look-Ahead.

4. Level Four – 4-Bit ALU

The 4-Bit ALU circuit integrated the 4-Bit Group GP, 4-Bit ALU GP, and 4-Bit Overflow circuits with the logic that will calculate the sums. The logic equations for the four sum bits were presented in Figure 11 and are realized in Figure 25.

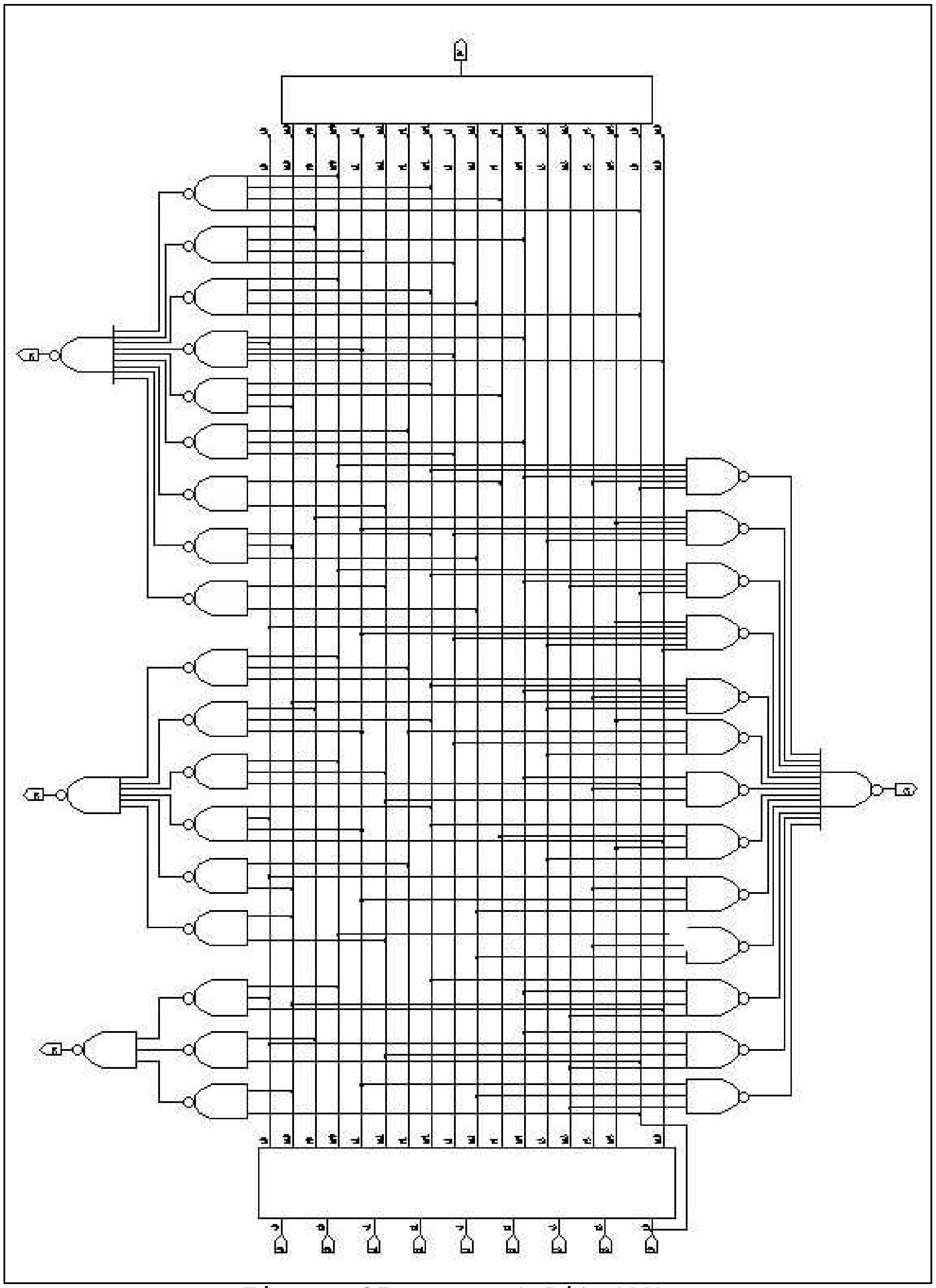


Figure 25.   4-Bit ALU.

5. Level Five - 16-Bit Adder

The 16-Bit Adder is assembled from four 4-Bit ALU and the Carry Look-Ahead (Figure 26).

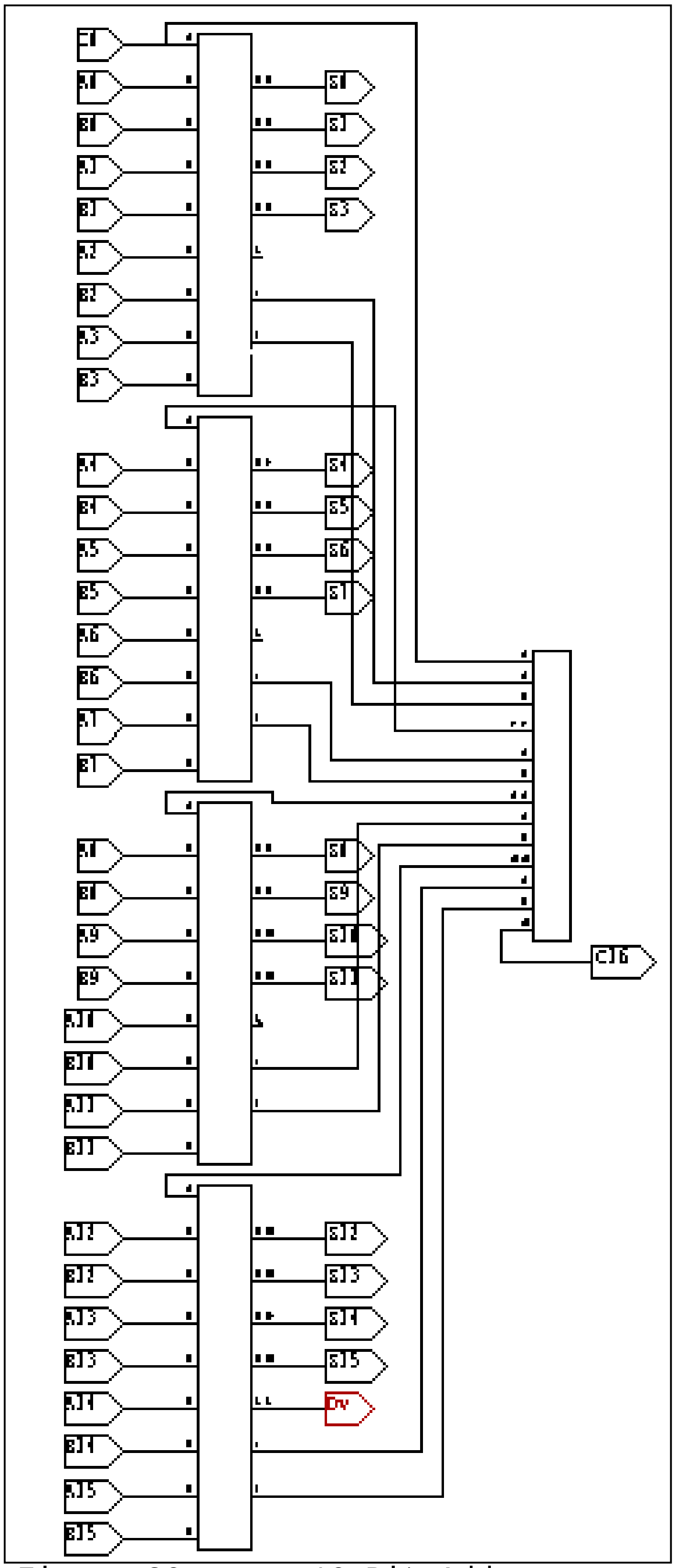


Figure 26.   16-Bit Adder.

C. SIMULATION OF SCHEMATIC DESIGN

SPICE net lists were created for all gates, sub-circuits and the 16-bit adder circuit. These were verified for electrical connectivity and logic functionality. Many of the sub-circuits underwent extensive functionality testing to confirm circuit completeness and provide an independent check that the logic equations were correct. Appendix A shows the net lists for all system circuits.

The 16-Bit Adder net list was tested for functionality and then subjected to exhaustive verification that its speed achieved the stated goal of 600 MHz. The register propagation time of 200 ps and skewing of input signals was simulated by a pulse shaping circuit of 4 inverters and having an input pulse ramp of 100 ps during low-to-high and high-to-low transitions. The SPICE net list header file (caa_Adderx16_625.h) included in Appendix C documents this effort.

In addition, recall that the schematic diagrams of the FETs included estimations of source and drain area (AS and AD) and perimter (PS and PD). This was an additional effort to make the simulations as realistic as possible, taking into account resistive and parasitic capacitive effects across the transistor gate.

IV. LAYOUT DESIGN OF 16-BIT ADDER

A. OVERVIEW

1. VLSI Fabrication Process

The stated task specified that the layout implementation of the 16-bit adder use the Taiwan Semiconductor Manufacturing Corporation (TSMC) 0.18 micron fabrication process. This process utilizes deep ultra-violet lithography with a lambda value of 0.09 um, resulting in a true minimum feature size of 0.18 microns for the gate lengths of the CMOS FETs. The TSMC 0.18 micron technology is a single poly, six metal layer process with low-k dielectrics. This process has a tight metal pitch, providing a higher gate density and more die per wafer, leading to a lower cost per chip. The process allows for stacked vias, permitting a dense, integrated circuit design. The deep sub-micron design rules allow densities of over 100,000 gates per mm^2 and on-chip clocking speeds of 600-700 MHz. Although a full line of standard cell libraries are available from various companies, this 16-bit adder layout doesn’t use any of them.

2. MOSIS

MOSIS is a non-profit, low-cost prototyping and small volume production service for VLSI integrated circuits. MOSIS works with a number of vendors for chip fabrication. MOSIS is the interface with Taiwan Semiconductor. MOSIS does not test the fabricated chips, however they do measure test structures and reference designs on each fabrication run to ensure it conforms to their standards. They also provide transistor simulation models and electrical properties of integrated circuit structures to be used in circuit simulators (T-Spice). The electrical properties for the TSMC 0.18 micron process from MOSIS are included in Appendix F.

3. L-Edit

The design process next turned to using the Tanner Tools layout editor (L-Edit) to construct the layout of the 16-bit adder from the schematic design. The layout of a design gives a representation of the lithography masks used for the chip fabrication process. L-Edit is CAD software used to generate a physical layout for the masks to be made by Taiwan Semiconductor. The first step in designing the integrated circuit mask layout was to set up the design file. This entailed defining the different layers to be used in the design, entering design rules for the fabrication process and defining the technology units to be used in the design. The layer setup assigned different colors and stipple patterns. Figure 27 details the layer colors and patterns used in the layout process.

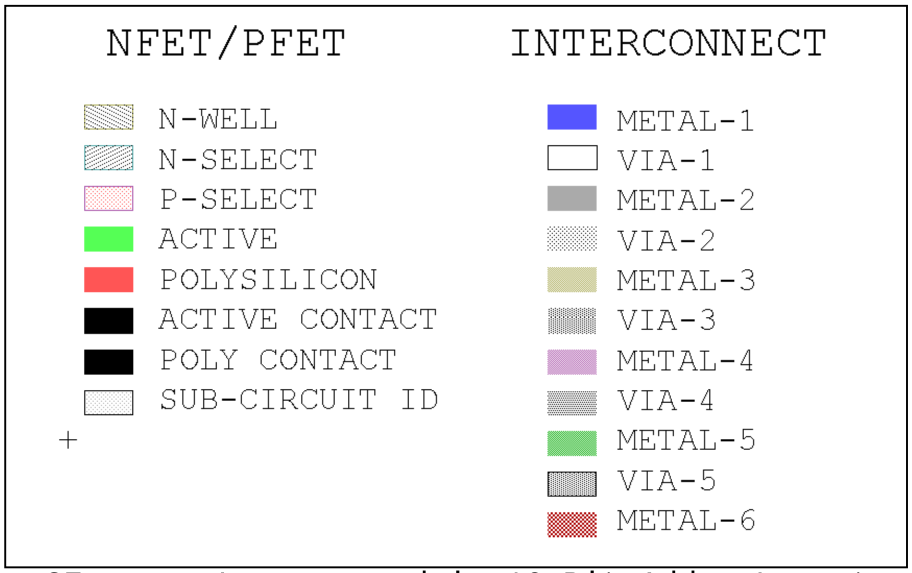


Figure 27. Layers used in 16-Bit Adder Layout.

This process generally followed the schematic design process in that it was hierarchical in nature. However, in a number of cases, sub-circuits were designed along a slightly different hierarchy. For example, the SPICE net list extractor for L-Edit can recognize combinations of active and poly as transistors and calculate their gate dimensions. Therefore, it was unnecessary to layout individual FETs and then combine them into gates. Thus, the mask layout discussed in the following paragraphs begins immediately with gate level design.

The mask layout experience gained in laying out the NOR gate adder (Figure 8) factored into the design approach of the rest of the layout. One of the difficulties encountered in the NOR gate layout was the fact that the fanout of the ALU Generate and Propagate functions was very large. The solution was to produce multiple copies of certain Generate and Propagate variables. The effect of this requirement was to make the mask layout inefficient.

The immediate benefit of the NAND design was that it did not have a fanout problem (Figure 11). The decision was made to design the heart of the 4-Bit ALU first, to include horizontal (left to right) Generate and Propagate variables. The summing circuits flank these variables, tapping off them. The ALU GP circuit and Group GP circuits are laid out to the left of the summing circuit and the overflow circuit is laid out to the right.

B. HIERARCHICAL LAYOUT DESIGN

1. Level One – Logic Gates

The logic gate layouts were designed to be placed horizontally next to each other, providing contiguous running Vdd and Gnd wires when assembled into sub-circuits. This enabled the ease in assembly of various sized gates into sub-circuits. For example, the 2-Input NAND Gate schematic from Figure 18 is realized for fabrication in Figure 28. See Appendix B for a complete listing of gate layout designs.

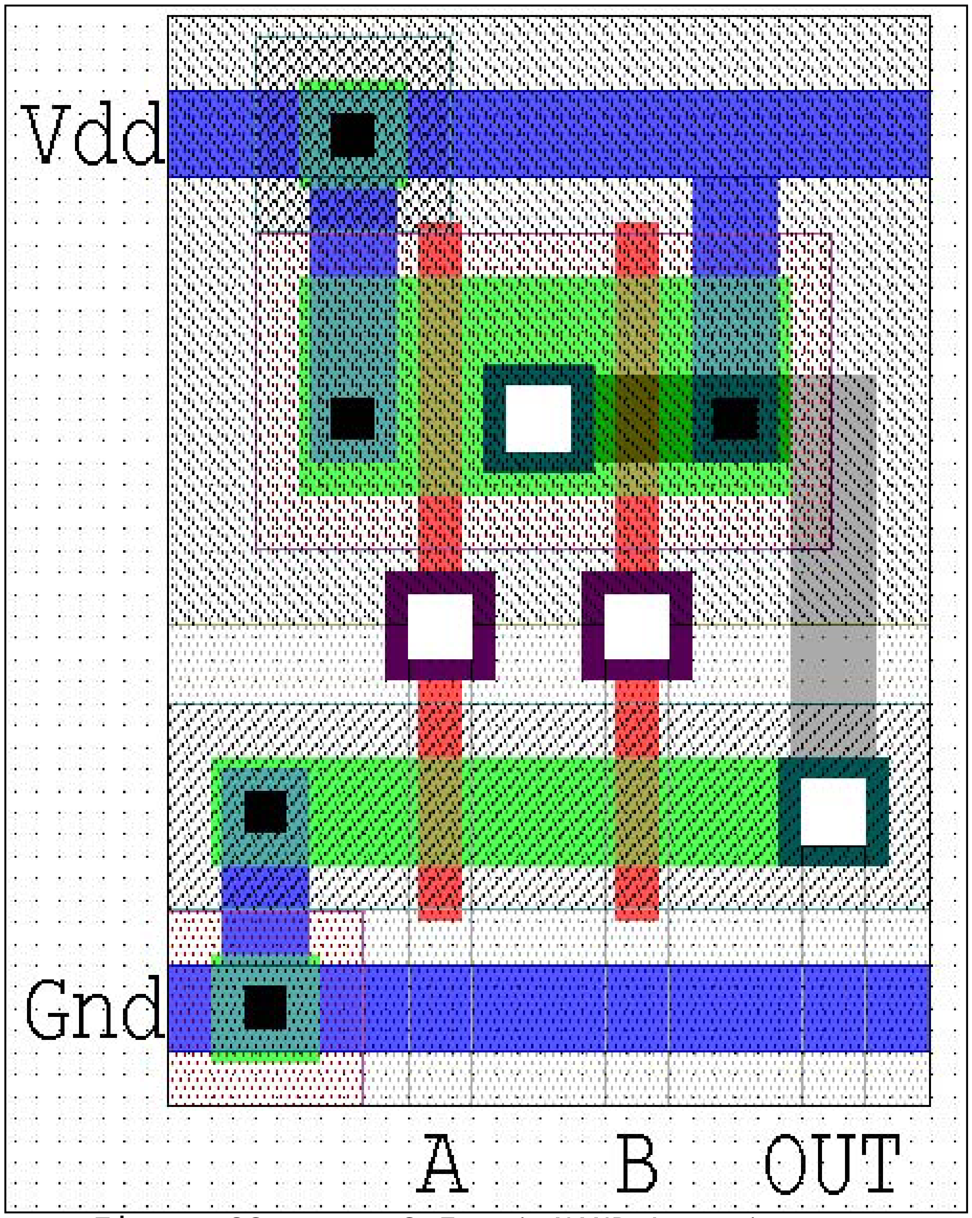


Figure 28. 2-Input NAND Layout.

2. Level Two – Secondary Sub-Circuits

a. Sum Circuits

The first sub-circuits produced were the sum circuits. The logic equations were presented in Figure 11 and their schematic design in Figure 25. At this point, they are devoid of specific input variables. When they are mated to the horizontal connections to the ALU Generate and Propagate functions, the logic equations will be realized. At this level of design, the task is to construct the internal circuitry and verify each sum circuit's electrical connectivity. The layout SPICE extractions for each circuit can be found in Appendix RR. The sum circuit layouts are presented in Figures 29 to 31.

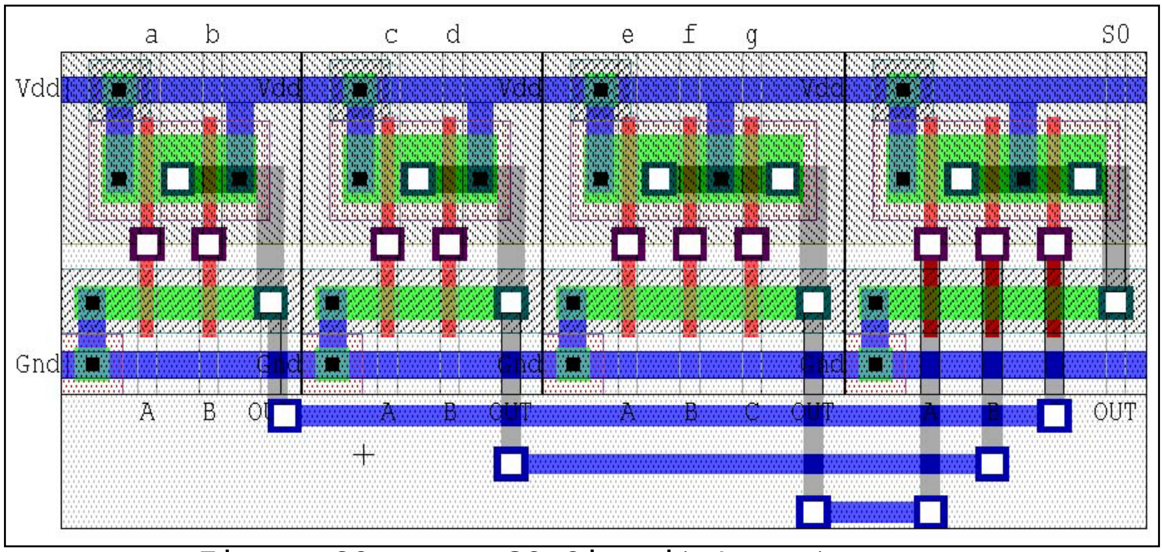


Figure 29. S0 Circuit Layout.

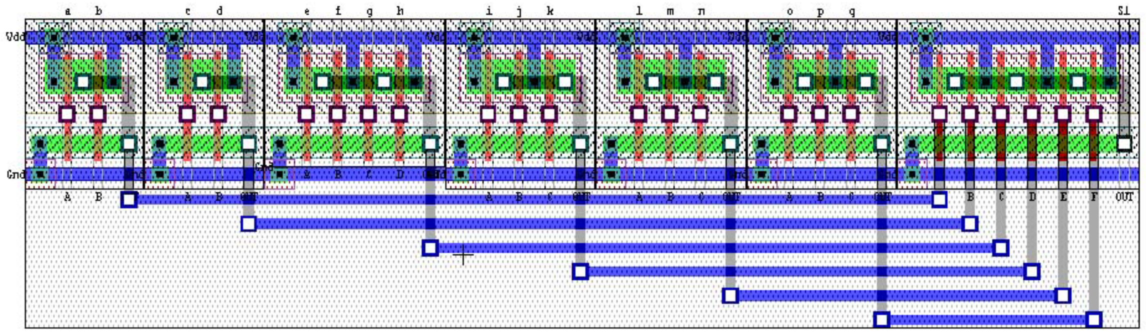


Figure 30. S1 Circuit Layout.

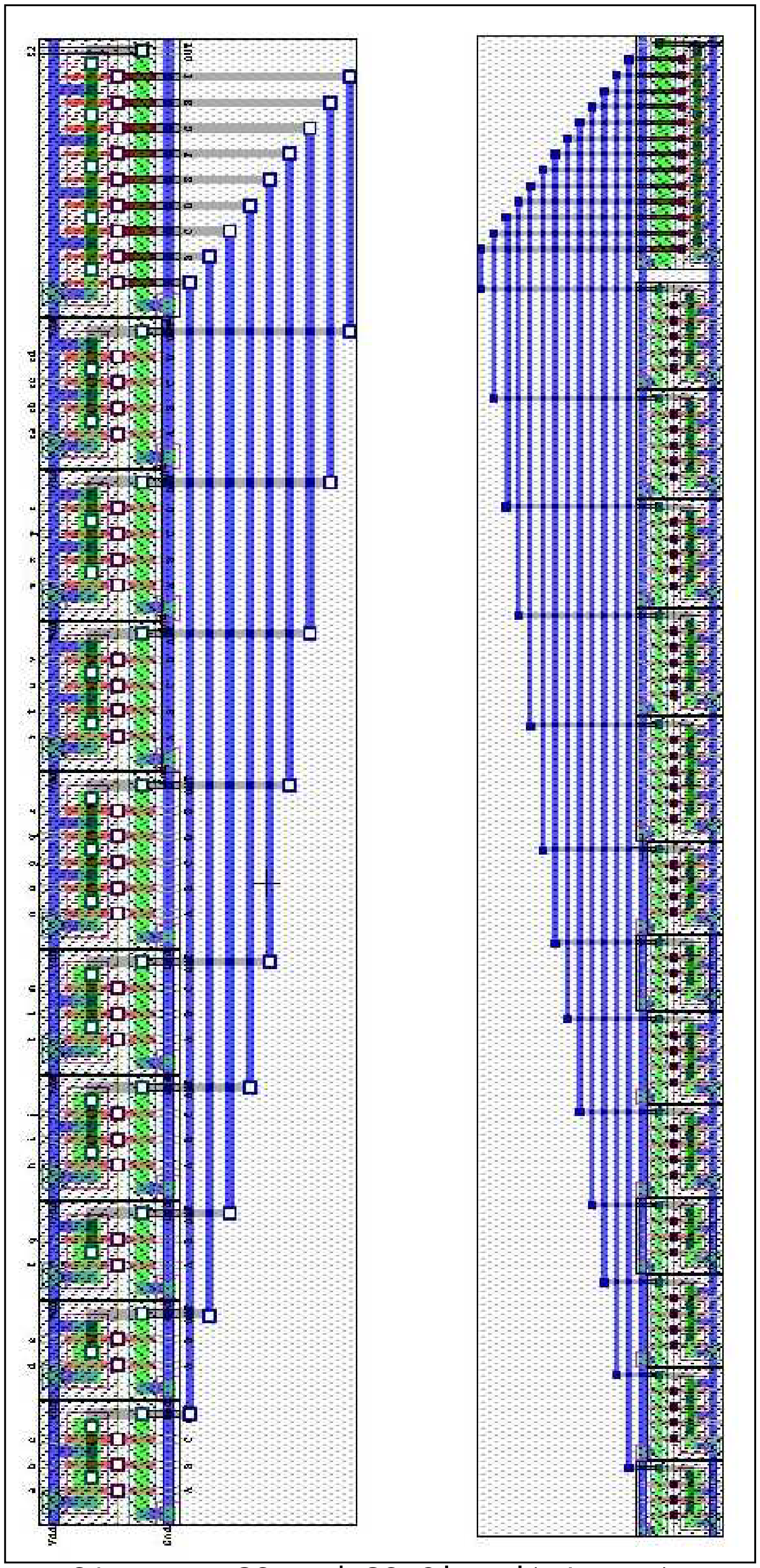


Figure 31. S2 and S3 Circuit Layouts.

b. 4-Bit ALU Group Generate and Propagate

The 4-Bit ALU Group Generate and Propagate circuit is physically located to the left of the 4-Bit ALU GP circuit in the 4-Bit ALU. It simultaneously uses the input bits A<3..0> and B<3..0> to calculate the Group Generate and Propagate for output to the Carry Look-Ahead Unit. The schematic design is presented in Figure 19. This layout was constructed later in the process and leverages some additional concepts and techniques of VLSI design. Note that there is little “wasted” space of just interconnect (Figure 32).

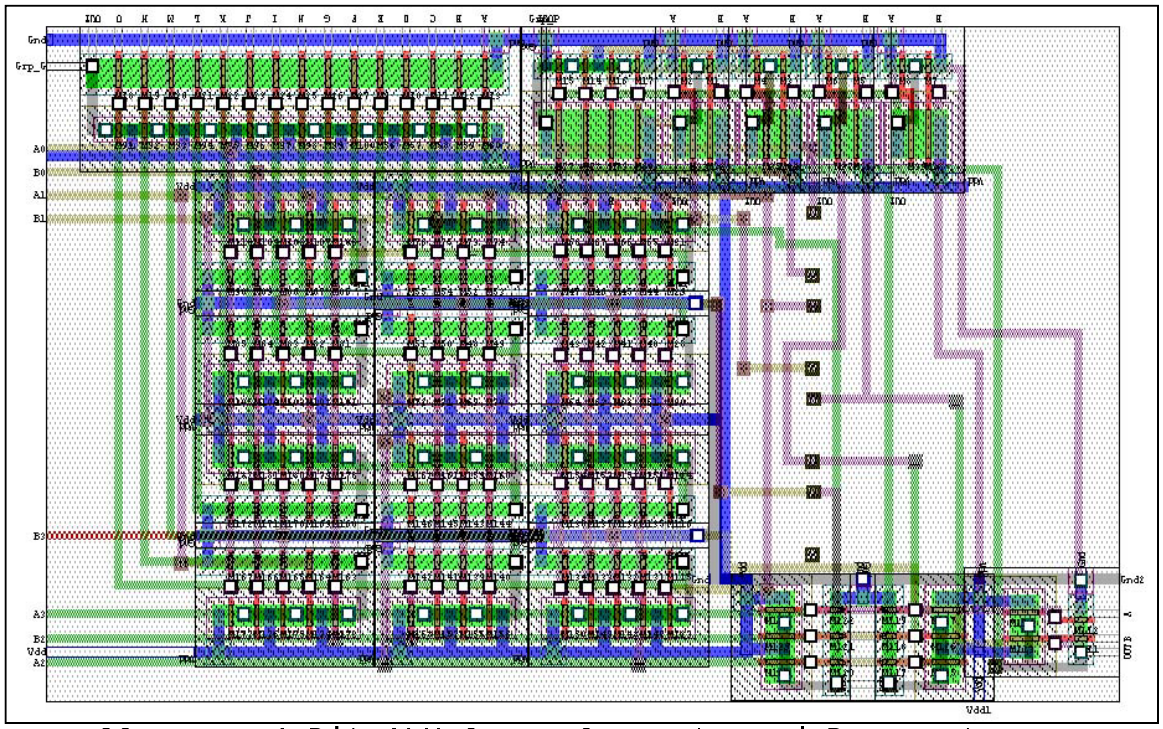


Figure 32. 4-Bit ALU Group Generate and Propagate.

The Grp_P output is realized in the upper right of the circuit. Note the 4-Input NOR in the middle of the upper row of gates that is processing the intermediate sums from the four 2-Input NOR gates to the right. The Grp_G output is realized by the 15-Input NAND located in the upper left of the circuit. The fifteen varying sizes of NAND gates that send it intermediate products are located in the middle and lower right of the circuit. The interconnects that run vertically in the empty space to the right will pass over the 4-Bit ALU GP circuit which is integrated in that space. The eight darker square pads into that empty space indicate where the A<3..0> and B<3..0> input will drop into the 4-Bit ALU GP circuit.

See Appendix B for the layouts of the combinational gate alternatives that were discussed above, Figures 20 and 21, but not used in the final layout.

c. 4-Bit ALU Generate and Propagate (ALU GP)

The schematic design for the ALU Generate and Propagate circuit is presented in Figure 20. In a like fashion to the process used in the schematic design, the 1-Bit ALU GP was constructed and then the 4-Bit ALU GP built from four instances of the 1-Bit ALU GP. The constraining factor for this design was the spacing of the Generate and Propagate (G&P) inputs into the sum circuits. The G&P inputs were run into the sum circuits on the Metal-3 layer with a separation of 5 lambda.

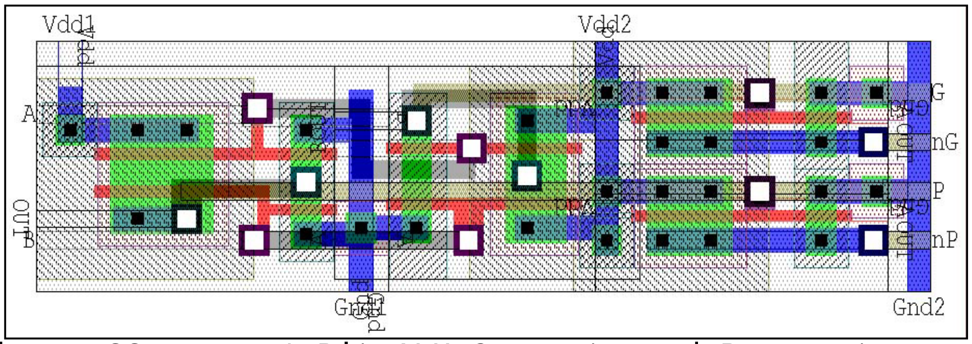


Figure 33. 1-Bit ALU Generate and Propagate.

In Figure 33, (from left to right) a 2-Input NAND is calculating the “Generate” (G) from the two inputs

A and B that are being input from the left. Next to that, there is a 2-Input NOR which is calculating the “Propagate” (P) from the two inputs. There are also two inverters stacked vertically. The topmost inverter is calculating the “Not-G” (nG) and the bottom inverter is calculating the “Not-P” (nP). The outputs (G, nG, P, nP) are leaving the circuit to the right.

The 4-Bit ALU GP consists of four instances of the 1-Bit ALU GP. In addition, there is one inverter at the top-right of the layout that calculates the “Not-C_0” (nC_0) (Figure 34).

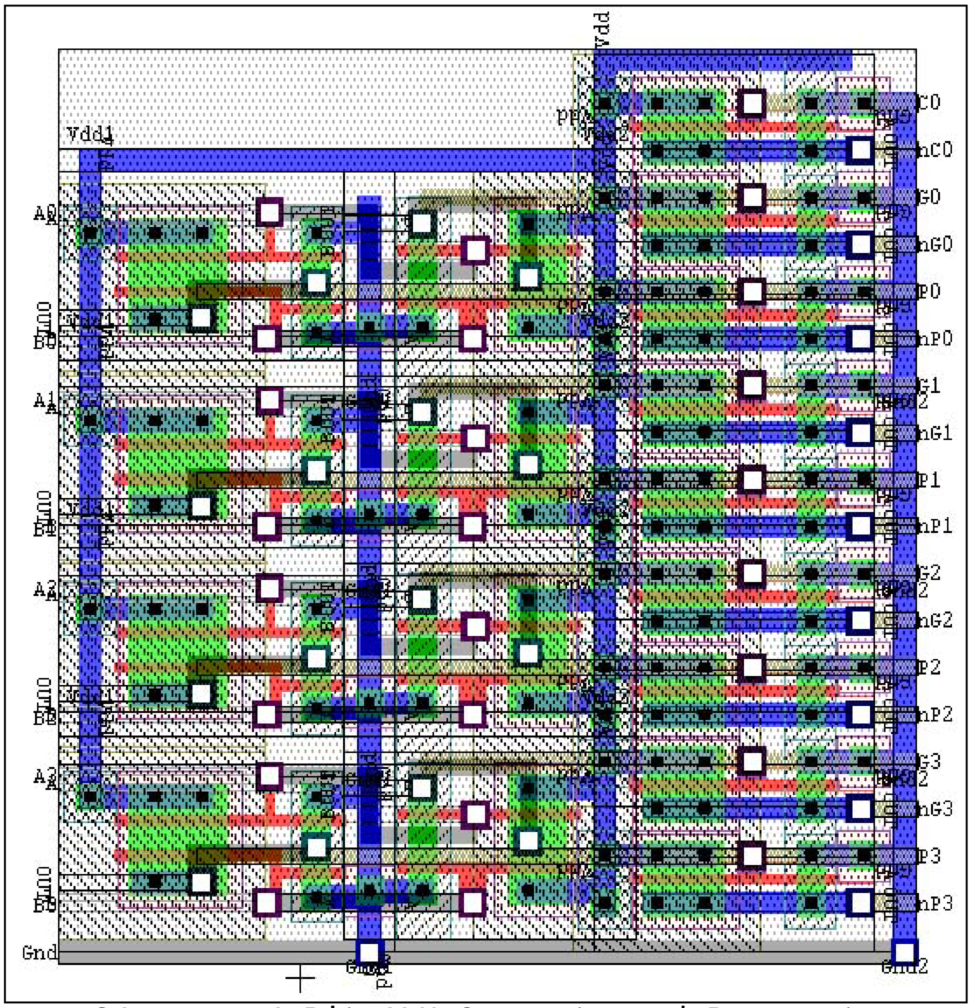


Figure 34. 4-Bit ALU Generate and Propagate.

d. 4-Bit ALU Overflow

The 4-Bit ALU Overflow circuit is designed to attach to the right side of the sum circuit. The design is also limited by the Generate and Propagate inputs that are being provided from the Sum circuit. The G&P interconnects have directed the dimensions of the 4-Bit ALU (Figure 35).

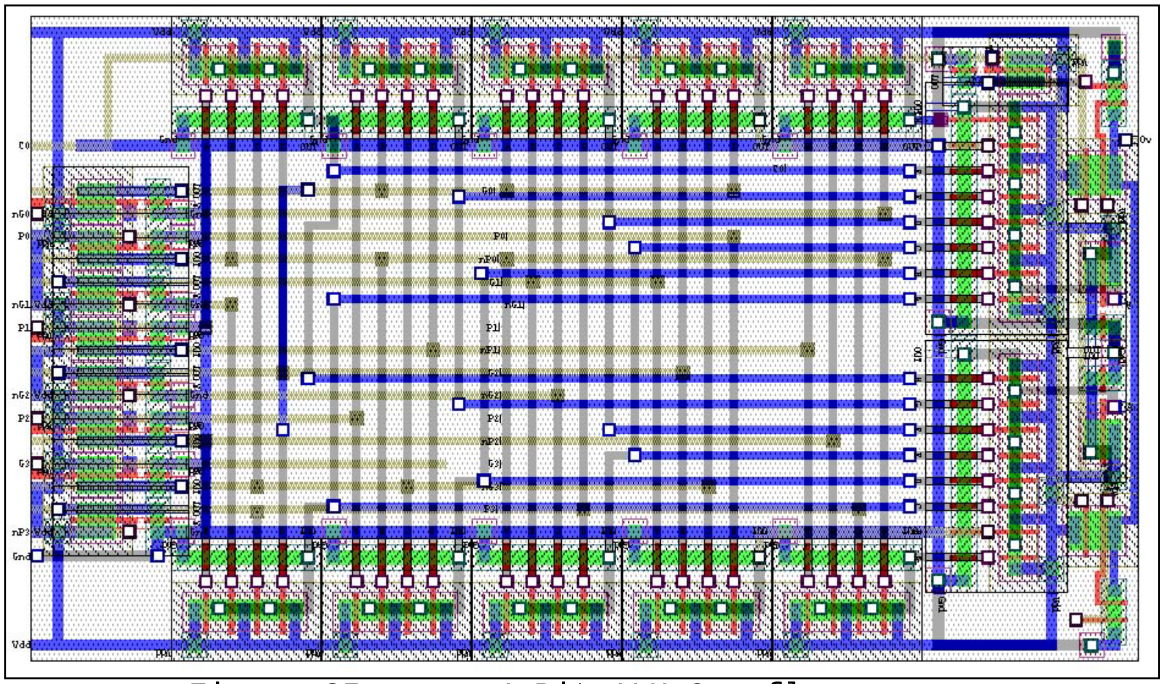


Figure 35. 4-Bit ALU Overflow.

3. Level Three – 4-Bit Adder Circuit

The sum circuits and other sub-circuits were assembled into the 4-Bit ALU (Figure 36). The inputs (A<3..0> and B<3..0>) come into the ALU from the left. They are used in the Group GP circuit to calculate Grp_G and Grp_P and then G<3..0> and P<3..0> in the ALU GP circuit. G<3..0> and P<3..0> travel horizontally to the right the length of the sum circuits to the overflow circuit. S<3..0> outputs exit the ALU to the right on Metal-6. The final signal calculated is overflow, which also exits on the right.

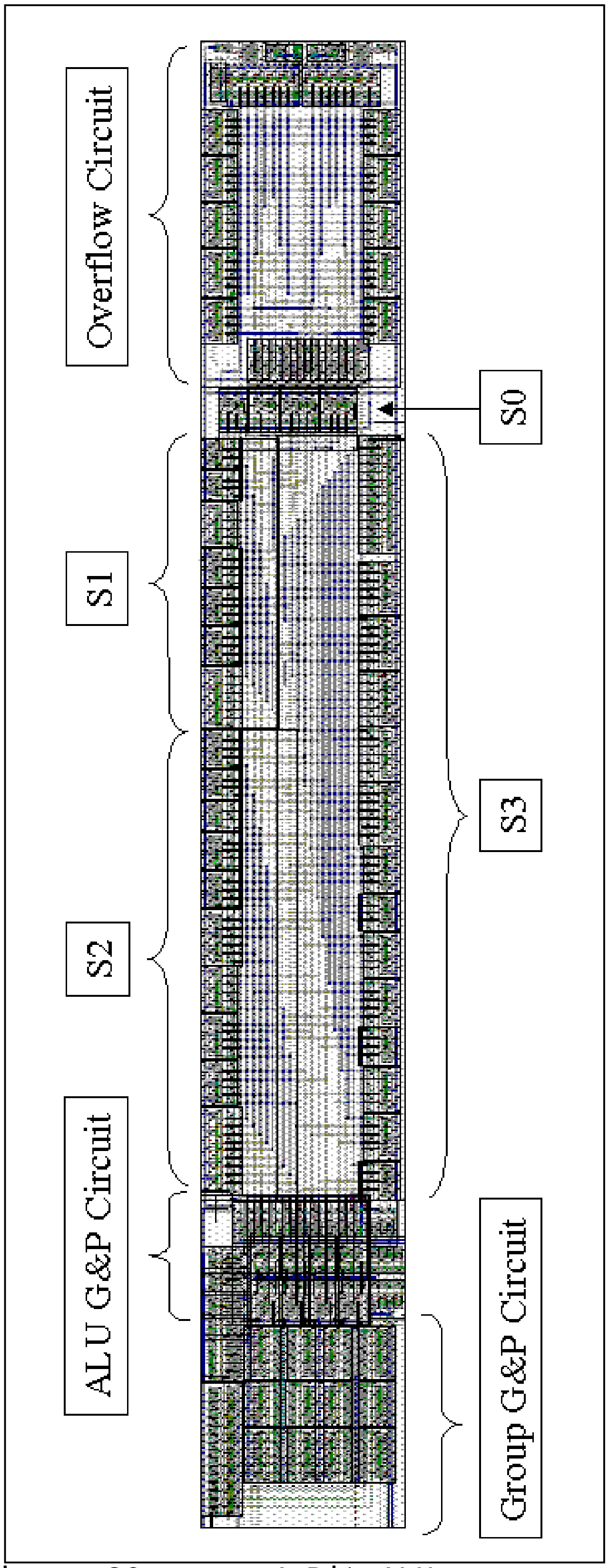


Figure 36. 4-Bit ALU.

4. Level Four – 16-Bit Adder

Four of the 4-bit ALU’s have been stacked vertically to form the 16-bit ALU block. The CLAH has been designed specifically to integrate into the ALU block to conserve area. This process is demonstrated in Figure 37.

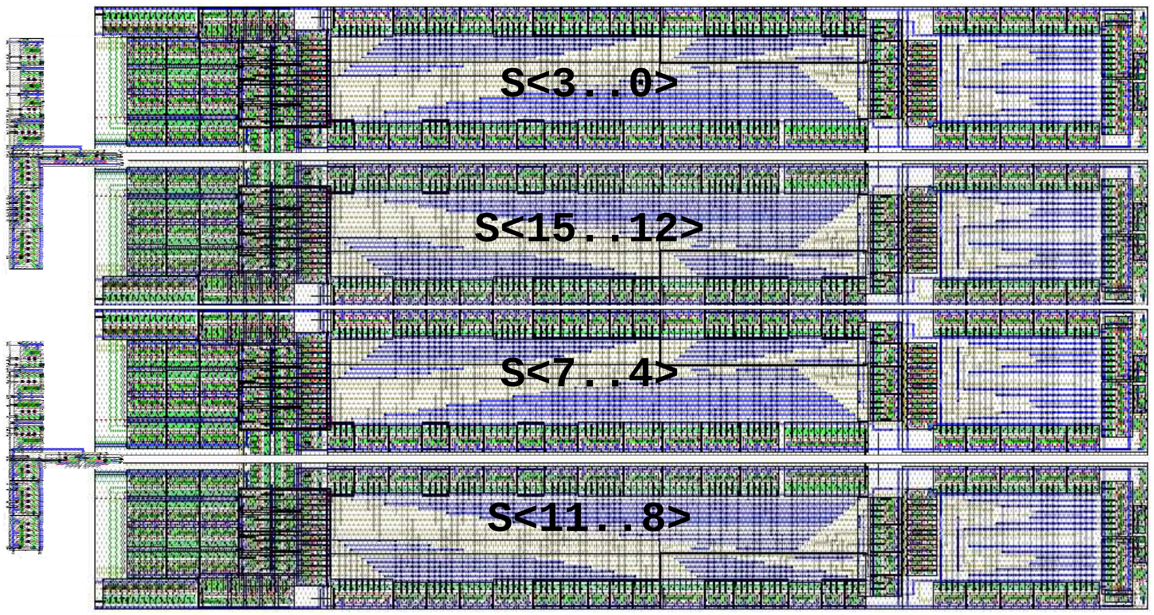


Figure 37. Integration of CLAH in 16-Bit Adder.

Recall that the logic equations used in the CLAH to calculate the upper level carries are of different sizes (Figures 13 and 22). It was not possible to have the 16-bit adder stacked sequentially from least significant bit to most significant bit. In order to realize a CLAH that would fit into the two ‘slots’ on the left of the 16-bit adder it was necessary to stagger the order. Figure 35 includes a notation for the sum bits produced by each 4-bit ALU. See Figure 38 to view the final 16-Bit Adder.

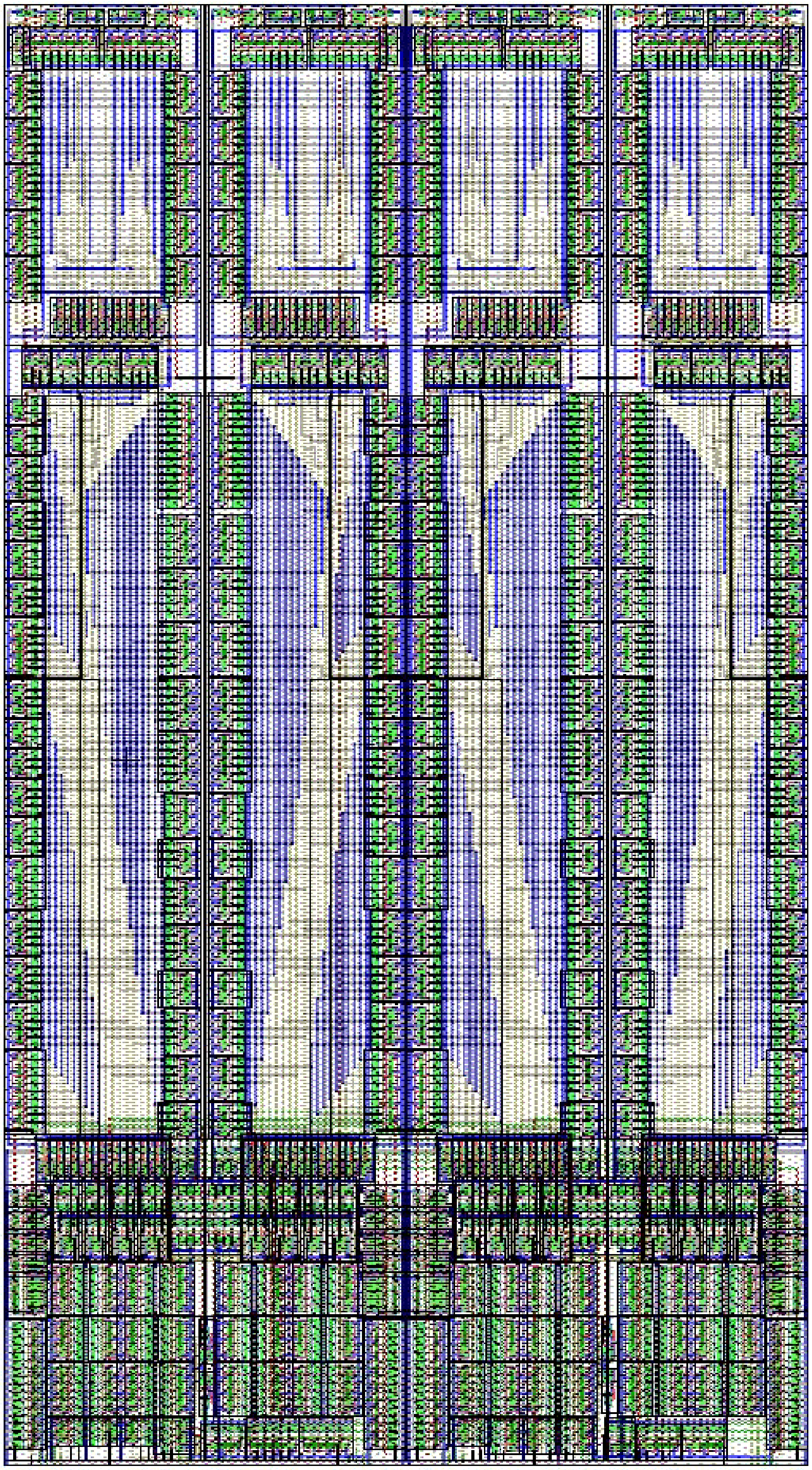


Figure 38. 16-Bit Adder.

C. LAYOUT VERSUS SCHEMATIC CHECKS

The fabrication mask layout was verified as being identical to the schematic design with the use of the Tanner Layout Versus Schematic (LVS) program. This program compares the SPICE net lists extracted from the layout and schematic design to ensure that they are identical. It ‘flattens’ the net lists to the transistor level, then verifies that all the transistors are the same and they are attached the same; i.e. node check.

Some difficulties were experienced with this program. It is important to have the program settings properly adjusted. The most significant problem encountered was how the LVS checker handled transistors in series. For example, the inputs into a 6-Input NAND in the schematic design may be in a different sequence than in the layout. Logically, they will accomplish the same function, but their nodes are different. There is a command in LVS to “compress” series transistors so this situation won’t lead to a node error.

Appendix G includes a pictorial tutorial recommending certain settings in order to facilitate the LVS check.

V. SUMMARY AND RECOMMENDATIONS

A. SUMMARY

The 16-Bit Adder w/Carry Look-Ahead described in this thesis is a robust design that functions well within the pipeline register architecture of the DIS. However, the design and a great amount of the layout were developed for a fabrication process utilizing a system voltage of 3.3 volts. It was discovered recently that the fabrication process had evolved to one utilizing a system voltage of 1.8 volts. The effect of this change was to slow the circuit from being able of meeting the stated goal of 600 MHz clock to 500 MHz clock speed. Additional design work has been conducted to achieve the 600 MHz clock speed. Recommendations are offered below for future work to modify the design to achieve 600 MHz.

The 16-Bit Adder layout occupies an area of 1600 x 900 lambda. For the 0.18 micron process, lambda equals 0.09 microns. Therefore, the area of one adder is 0.0117 mm^2. The total area used by the 1024 adders for the 512 range bin DIS will be 12 mm^2. In addition, it was determined that the adder has an average current consumption of 0.275 mA. Therefore, the average power consumed by the 512 range bin DIS will be 0.5 watts.

$$Avg\ Power = iV = 1024 \times 0.275e-3 \times 1.8 = 0.5W \quad (5)$$

The design offers a number of capabilities that may be exploited. The primary one is that although it was designed and implemented for fabrication as part of an ASIC, it can readily be implemented as part of a field programmable gate array (FPGA) design. The NAND gate logic can be quickly converted into a product-of-sums format that is efficiently implemented in an FPGA.

A second capability to note is that although it has been designed to be fully functional within a single stage, it can be easily pipelined. This may become necessary if the fabrication process changes (again), slowing the adder, or the requirements change and the process must be accomplished with a faster clock. For a two-stage adder, the recommendation would be to have the Group GP, ALU GP, and CLAH circuits in the first stage and the 4-Bit ALU and Overflow circuits in the second stage. Besides the necessity to have another level of registers, this two-stage pipeline has no additional ALU circuits to be designed.

B. RECOMMENDATIONS

Schematic design for the following design modifications has been done and testing conducted within the actual register structure of the pipelined VLSI design. The results of this realistic testing are very encouraging as the modified 16-Bit Adder design was found to be capable of operating with a 625 Mhz clock speed. Details of the design modifications described below and the circuit simulator are presented in Appendix H.

1. 16-Bit Overflow Circuit Versus 4-Bit Overflow

A decision was made to design a 4-Bit Overflow circuit so as to make the 4-Bit ALU a fully capable unit. This would provide flexibility in the future if the design requirements were to change, or it became necessary to have a 4-Bit ALU w/overflow detection. However, when the 4-Bit

Overflow fabrication layout was done, it became apparent that it is significant in size [35]. This circuit has 142 transistors. Since only one overflow circuit is being utilized in the 16-bit Adder, eliminating the 4-Bit Overflow from the 4-Bit ALU, and designing an overflow circuit for the 16-Bit Adder (16-Bit Overflow) would save 426 (142 x 3) transistors per instance of 16-Bit Adder. For the DIS, with 1024 instances of the 16-Bit Adder, that's a savings of 436,224 transistors!

2. Multiplexing Sum Circuits with C_0

It became apparent that the sum outputs, particularly S2 and S3, of the 4-Bit ALU were the 'long poles in the tent' toward reducing the propagation time of the 16-Bit Adder. Experiments in implementing combinational gates, vice logic terms, were found to be ineffective (Figures 20 and 21). However, the multiplexing technique used in the 4-Bit Overflow circuit was very effective in that case where all of the inputs, except for C_0, were available early. This technique was applied to the sum circuit found to be very advantageous (Appendix H). Preliminary simulations indicate that an additional 0.2-0.3 ns can be saved from the propagation time of S2 and S3 by calculating two signals, Cis0 and Cis1, and multiplexing them with C_0.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Jane’s Information Group, “Airborne Surveillance, Maritime Patrol and Navigation Radars, United States of America,” [http://online.janes.com/search97cgi/s97_cgi?action=View&VdkVgwKey=%2Fcontent1%2Fjanesdata%2Fyb%2Fjrew%2Fjrew0504%2Ehtm&DocOffset=3&DocsFound=68&QueryZip=isar%2C+missle&Collection=current%5FMags&Collection=current%5FSent&Collection=current%5FDir&Collection=current%5FSRep&Collection=current%5Fyb&Collection=current%5FImagelib&ViewTemplate=janes%5Fdoc%5Fview%2Ehts&Prod_Name=JREW&], 11 October 2000.

2. Jane’s Information Group, “Airborne Surveillance, Maritime Patrol and Navigation Radars, Russian Federation and Associated States (CIS),” [http://online.janes.com/search97cgi/s97_cgi?action=View&VdkVgwKey=%2Fcontent1%2Fjanesdata%2Fyb%2Fjrew%2Fjrew1646%2Ehtm&DocOffset=1&DocsFound=3156&QueryZip=sea+dragon%2C+cis&Collection=current%5FMags&Collection=current%5FSent&Collection=current%5FDir&Collection=current%5FSRep&Collection=current%5Fyb&Collection=current%5FImagelib&ViewTemplate=janes%5Fdoc%5Fview%2Ehts&Prod_Name=JREW&], 2 July 2001.

3. Pace, P. E., Burton, G. D., “Anti-ship Cruise Missiles: Technology, Simulation and Ship Self-Defense,” *Journal of Electronic Defense*, Vol. 21, No. 11, pp. 51-56, Nov 1998.

4. Pace, P. E., Fouts, D. J., Ekestorm, S., Karow, C., “A Digital False-Target Image Synthesizer for Countering ISAR,” *IEEE Transactions on Aerospace and Electronic Systems*, pp. 1-4, in review.

5. Miller, R. L., Nothnick, C. E., and Bailey, D. S., Acoustic Charge Transport: Device Technology and Applications, *Microwave Library*, May 1992.

6. Yoon, C. H., Kim, H., and Shim, J. D., “A Tunable Optical Add/Drop Multiplexer Using a Fiber-Optic Tapped Delay-Line Transversal Filter,” *Pacific Rim Conference on Lasers and Electro-Optics*, Vol. 3, pp. 785-786, Sep 1999.

7. D. J. Fouts, P. E. Pace, C. Karow, S. Ekestorm, "A Single-Chip False Target Radar Image Generator for Countering Wideband Imaging Radars," *IEEE Journal on Solid State Circuits*, pg. 13, in review.

8. Tanner Research, Inc., “Tanner Tools Pro,” [http://www.tanner.com/EDA/products/tannertoolspro/default.htm], undtd.

9. Boyce, D. E., “LASI Integrated Circuit CAD for Windows,” [http://members.aol.com/lasicad/]

10. The MathWorks, Inc., “MATLAB V5.3,” [http://www.mathworks.com/]

11. MOSIS, “*MOSIS Process Information*,” [http://www.mosis.edu/Technical/Processes/proc-tsmc-cmos018.html], undtd.

12. Wakerly, J. F., *Digital Design Principles and Practices*, Third Edition, pp. 436-443, Prentice Hall, 2000.

13. Roth, Jr., C. H., *Fundamentals of Logic Design, Third Edition*, pp. 141-161, West Publishing Company, 1985.

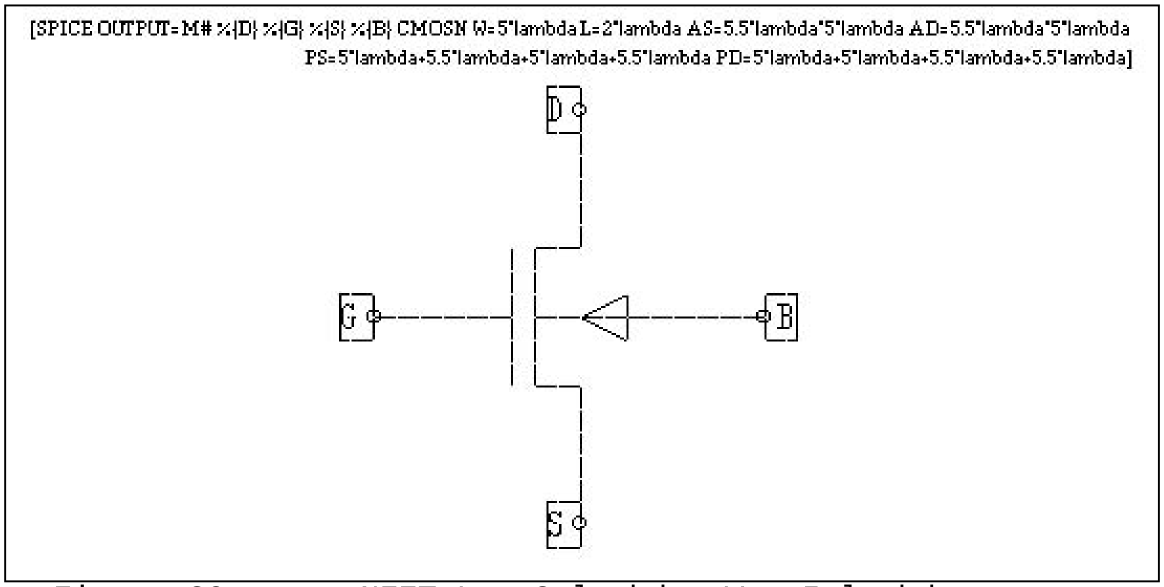
14. Intel, Corp, “Mobile Intel Pentium III Processor in BGA2 and PGA2 Packages Datasheet,” [ftp://download.intel.com/design/mobile/datashts/p3_ds.pdf], undtd.

APPENDIX A. SCHEMATIC DIAGRAMS W/SPICE NET LISTS

A. LEVEL ONE - FIELD EFFECT TRANSISTORS (FET)

1. NFET's

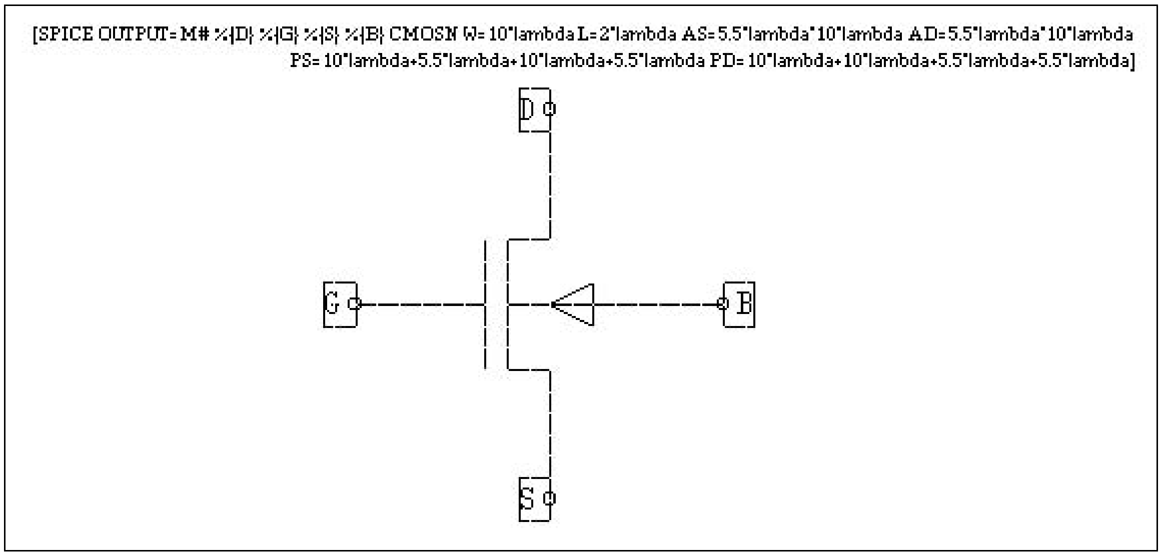
a. Length = 2 Lambda, Width = 5 Lambda



```
[SPICE OUTPUT=M# %{D} %{G} %{S} %{B} CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
        PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda]
```

Figure 39. NFET L = 2 lambda, W = 5 lambda.

b. Length = 2 Lambda, Width = 10 Lambda

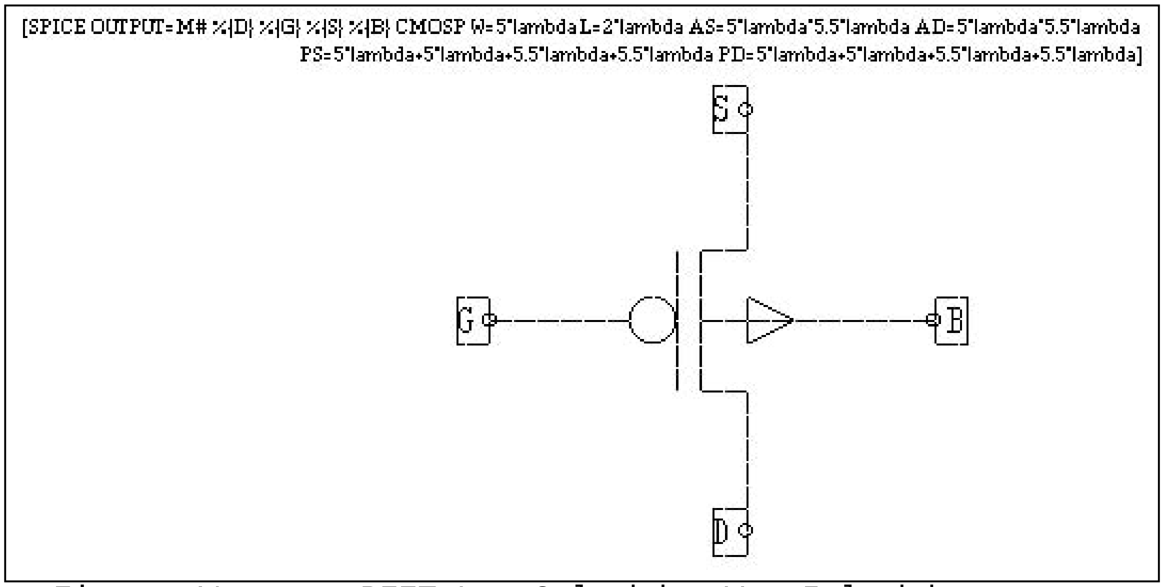


```
[SPICE OUTPUT=M# %{D} %{G} %{S} %{B} CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
        PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda]
```

Figure 40. NFET L = 2 lambda, W = 10 lambda.

2. PFET’s

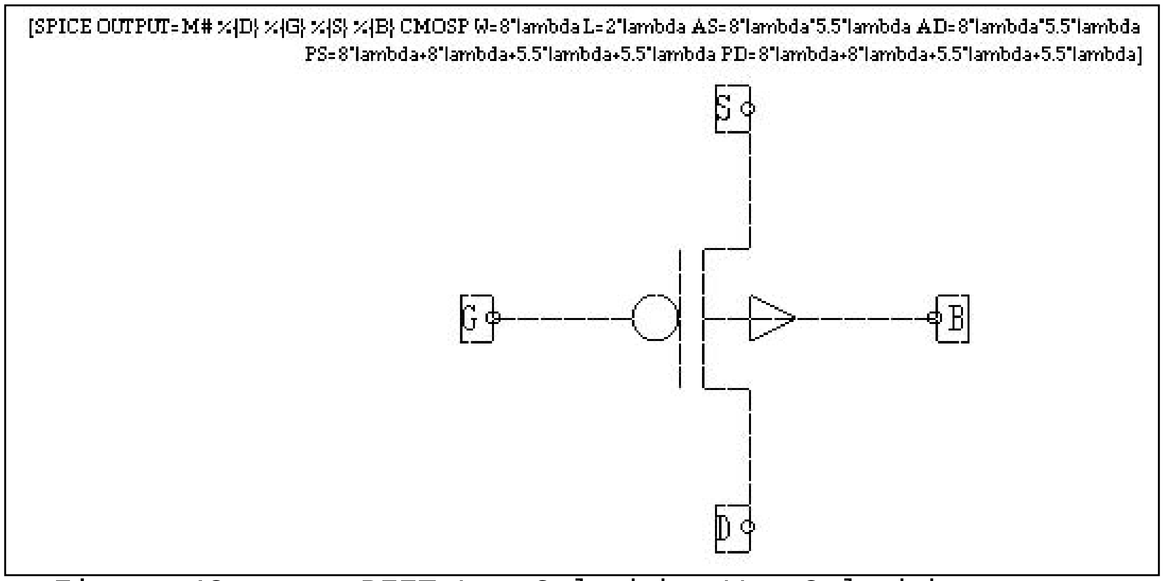
a. Length = 2 Lambda, Width = 5 Lambda



```
[SPICE OUTPUT=M# %{D} %{G} %{S} %{B} CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda]
```

Figure 41. PFET L = 2 lambda, W = 5 lambda.

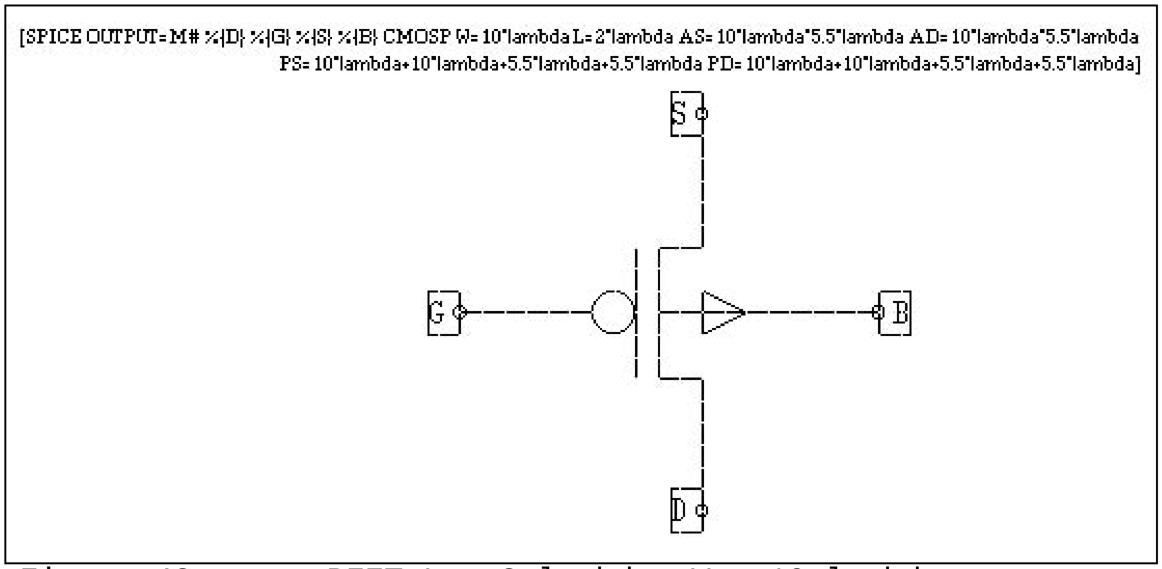
b. Length = 2 Lambda, Width = 8 Lambda



```
[SPICE OUTPUT=M# %{D} %{G} %{S} %{B} CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda]
```

Figure 42. PFET L = 2 lambda, W = 8 lambda.

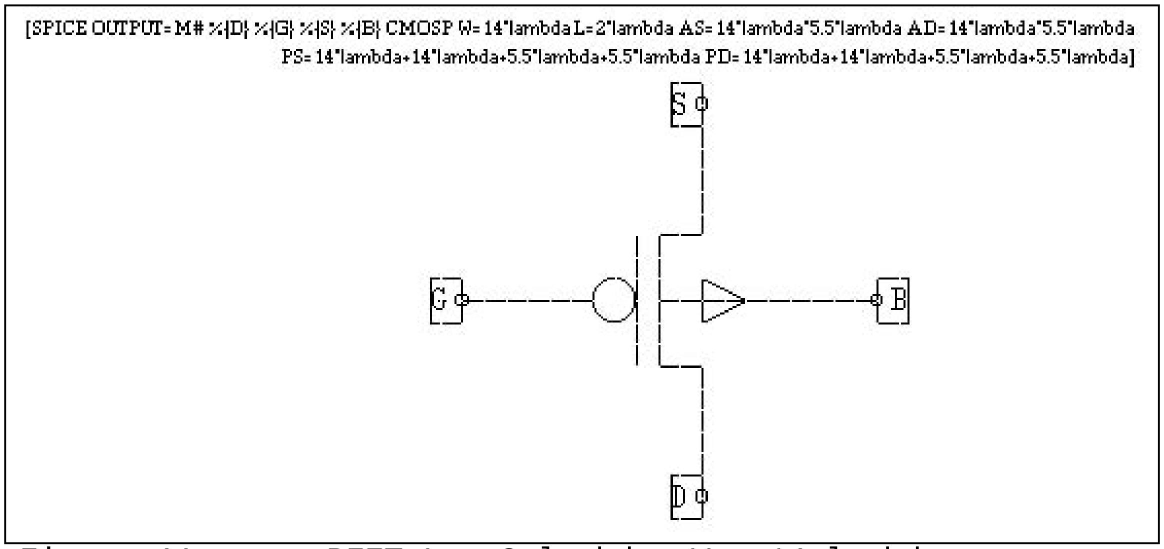
c.   Length = 2 Lambda, Width = 10 Lambda



```
[SPICE OUTPUT= M# %{D} %{G} %{S} %{B} CMOSP W= 10*lambda L= 2*lambda AS= 10*lambda*5.5*lambda AD= 10*lambda*5.5*lambda
PS= 10*lambda+10*lambda+5.5*lambda+5.5*lambda PD= 10*lambda+10*lambda+5.5*lambda+5.5*lambda]
```

Figure 43.   PFET L = 2 lambda, W = 10 lambda.

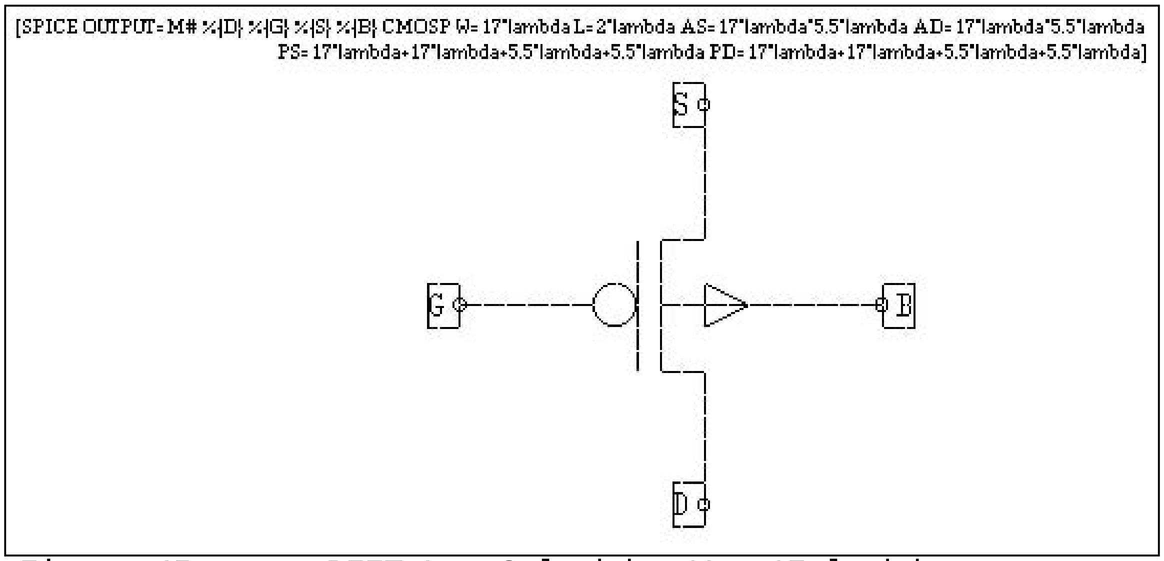
d.   Length = 2 Lambda, Width = 14 Lambda



```
[SPICE OUTPUT= M# %{D} %{G} %{S} %{B} CMOSP W= 14*lambda L= 2*lambda AS= 14*lambda*5.5*lambda AD= 14*lambda*5.5*lambda
PS= 14*lambda+14*lambda+5.5*lambda+5.5*lambda PD= 14*lambda+14*lambda+5.5*lambda+5.5*lambda]
```

Figure 44.   PFET L = 2 lambda, W = 14 lambda.

e. Length = 2 Lambda, Width = 17 Lambda



```
[SPICE OUTPUT= M# %{D} %{G} %{S} %{B} CMOSP W= 17*lambda L= 2*lambda AS= 17*lambda*5.5*lambda AD= 17*lambda*5.5*lambda
PS= 17*lambda+17*lambda+5.5*lambda+5.5*lambda PD= 17*lambda+17*lambda+5.5*lambda+5.5*lambda]
```

Figure 45. PFET L = 2 lambda, W = 17 lambda.

B. LEVEL TWO – LOGIC GATES

1. Inverter

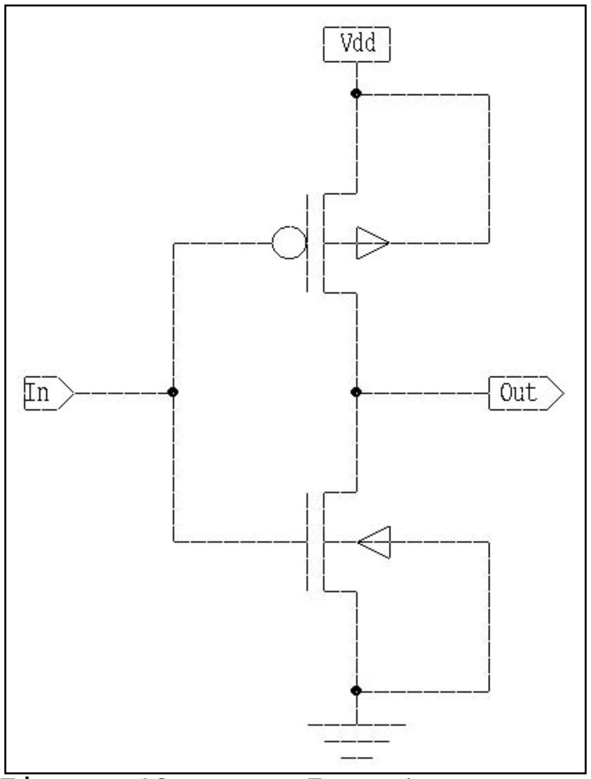


Figure 46. Inverter.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 14, 2001 at 14:48:35

* Main circuit: caa_INV
M1 Out In Gnd Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 Out In Vdd Vdd CMOSP W=14*lambda L=2*lambda
     AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
     PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
     PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_INV
.END
```

2. NAND

a. 2-Input

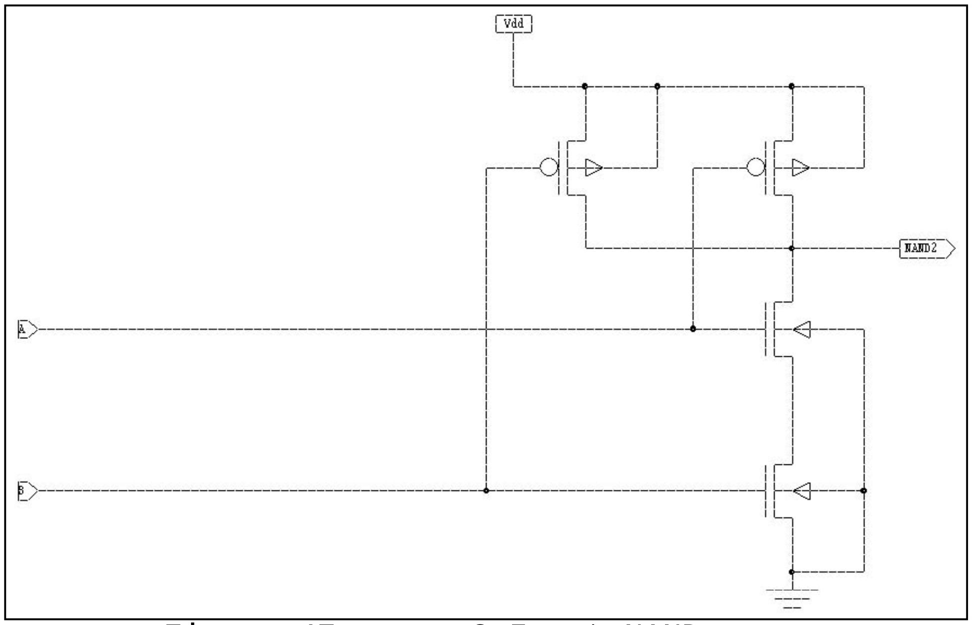


Figure 47. 2-Input NAND.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 15, 2001 at 10:52:56
* Main circuit: caa_NAND2
M1 NAND2 A N1 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B Gnd Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 NAND2 B Vdd Vdd CMOSP W=10*lambda L=2*lambda
     AS=10*lambda*5.5*lambda AD=10*lambda*5.5*lambda
     PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
     PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M4 NAND2 A Vdd Vdd CMOSP W=10*lambda L=2*lambda
     AS=10*lambda*5.5*lambda AD=10*lambda*5.5*lambda
     PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
     PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND2
.END
```

b. 3-Input

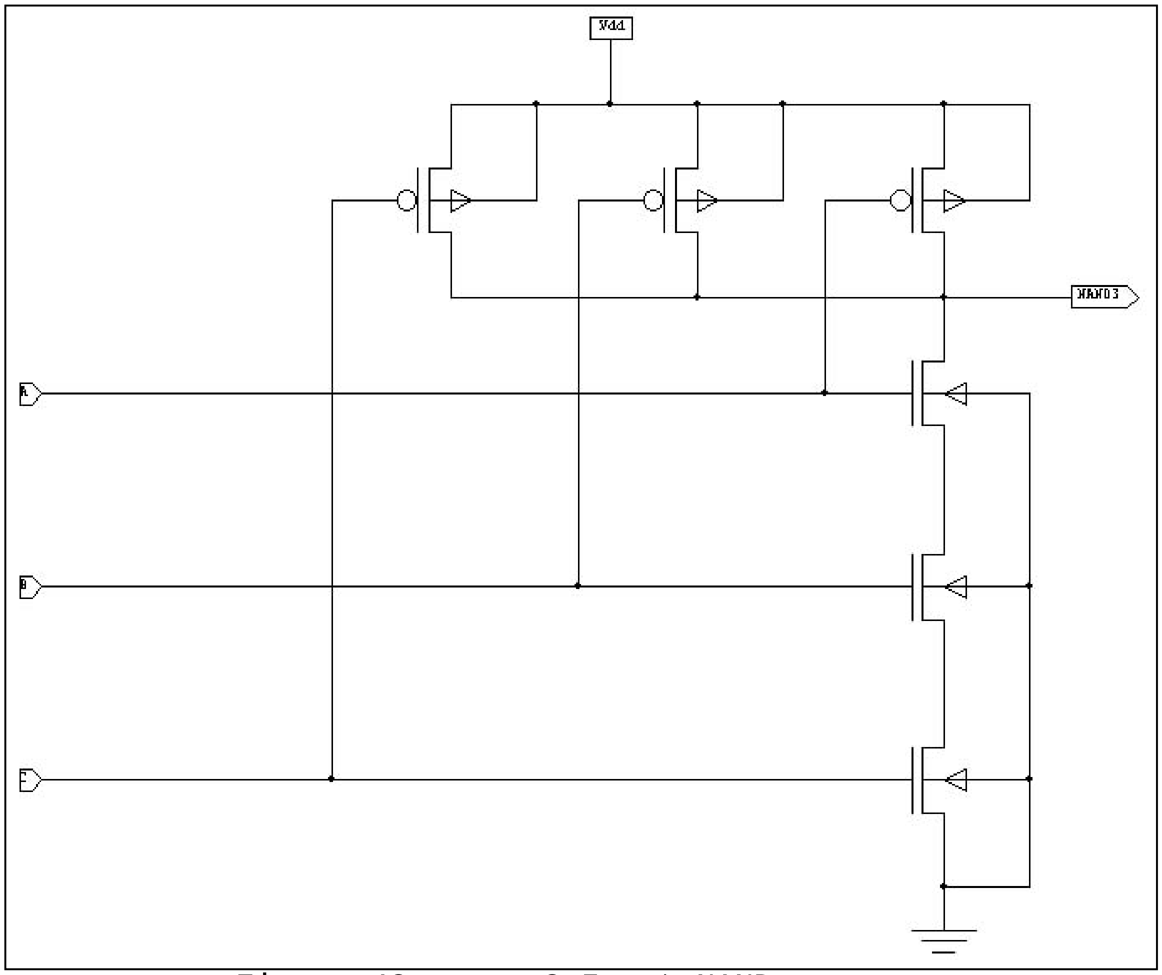


Figure 48. 3-Input NAND.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 15, 2001 at 09:22:46

* Main circuit: caa_NAND3
M1 NAND3 A N2 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N2 B N3 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N3 C Gnd Gnd CMOSN W=5*lambda L=2*lambda
```

```
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 NAND3 C Vdd Vdd CMOSP W=10*lambda L=2*lambda
     AS=10*lambda*5.5*lambda AD=10*lambda*5.5*lambda
     PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
     PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M5 NAND3 B Vdd Vdd CMOSP W=10*lambda L=2*lambda
     AS=10*lambda*5.5*lambda AD=10*lambda*5.5*lambda
     PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
     PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M6 NAND3 A Vdd Vdd CMOSP W=10*lambda L=2*lambda
     AS=10*lambda*5.5*lambda AD=10*lambda*5.5*lambda
     PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
     PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND3
.END
```

c. 4-Input

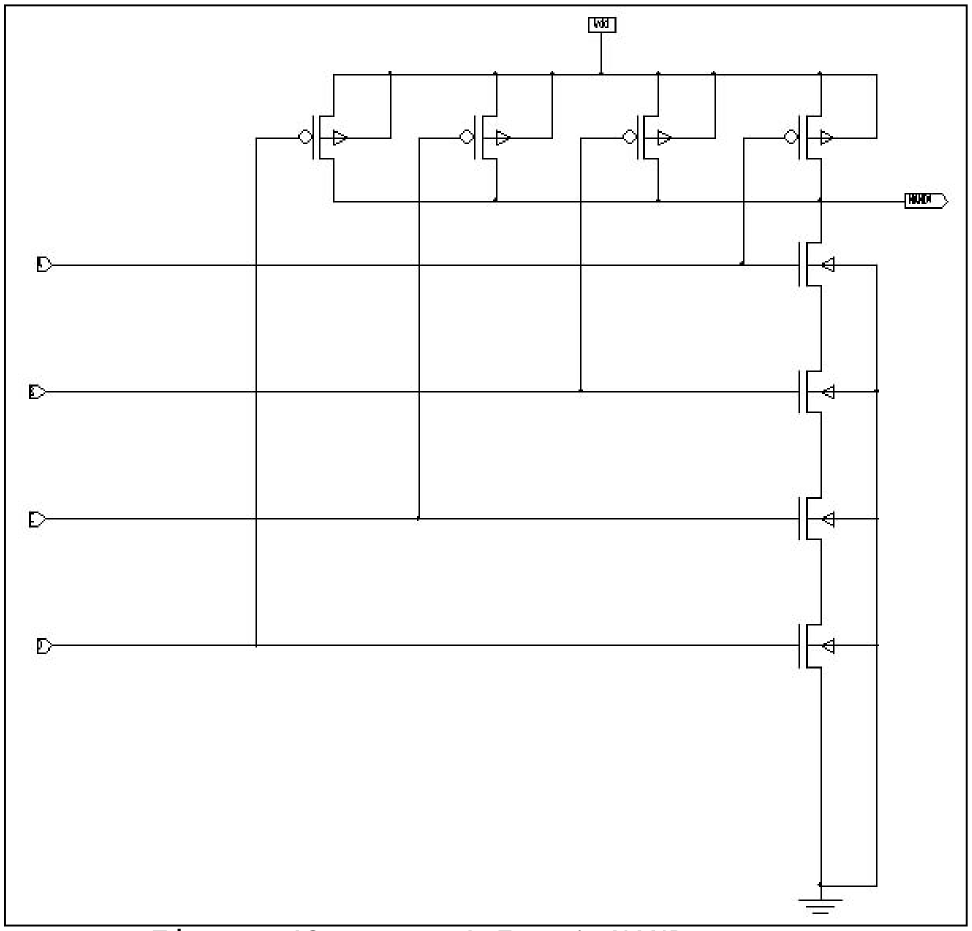


Figure 49. 4-Input NAND.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 15, 2001 at 09:37:19

* Main circuit: caa_NAND4
M1 NAND4 A N4 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N4 B N2 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N2 C N1 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N1 D Gnd Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 NAND4 A Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M6 NAND4 B Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M7 NAND4 C Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M8 NAND4 D Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND4
.END
```

d. 5-Input

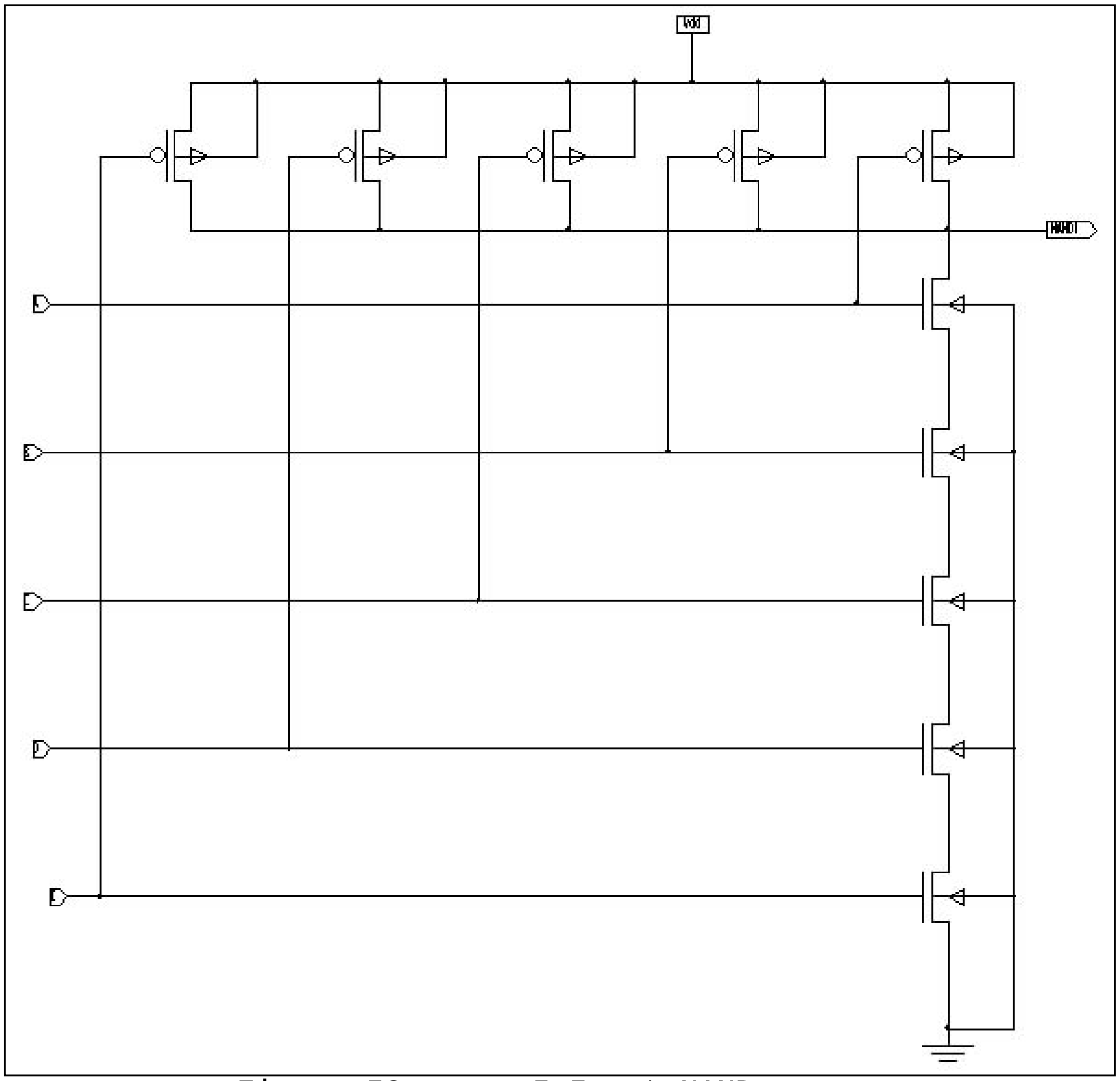


Figure 50. 5-Input NAND.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 15, 2001 at 09:37:50

* Main circuit: caa_NAND5
M1 NAND5 A N1 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B N5 Gnd CMOSN W=5*lambda L=2*lambda
```

```
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C N8 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N8 D N11 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N11 E Gnd Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 NAND5 A Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M7 NAND5 B Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M8 NAND5 C Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M9 NAND5 D Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M10 NAND5 E Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND5
.END
```

e. 6-Input

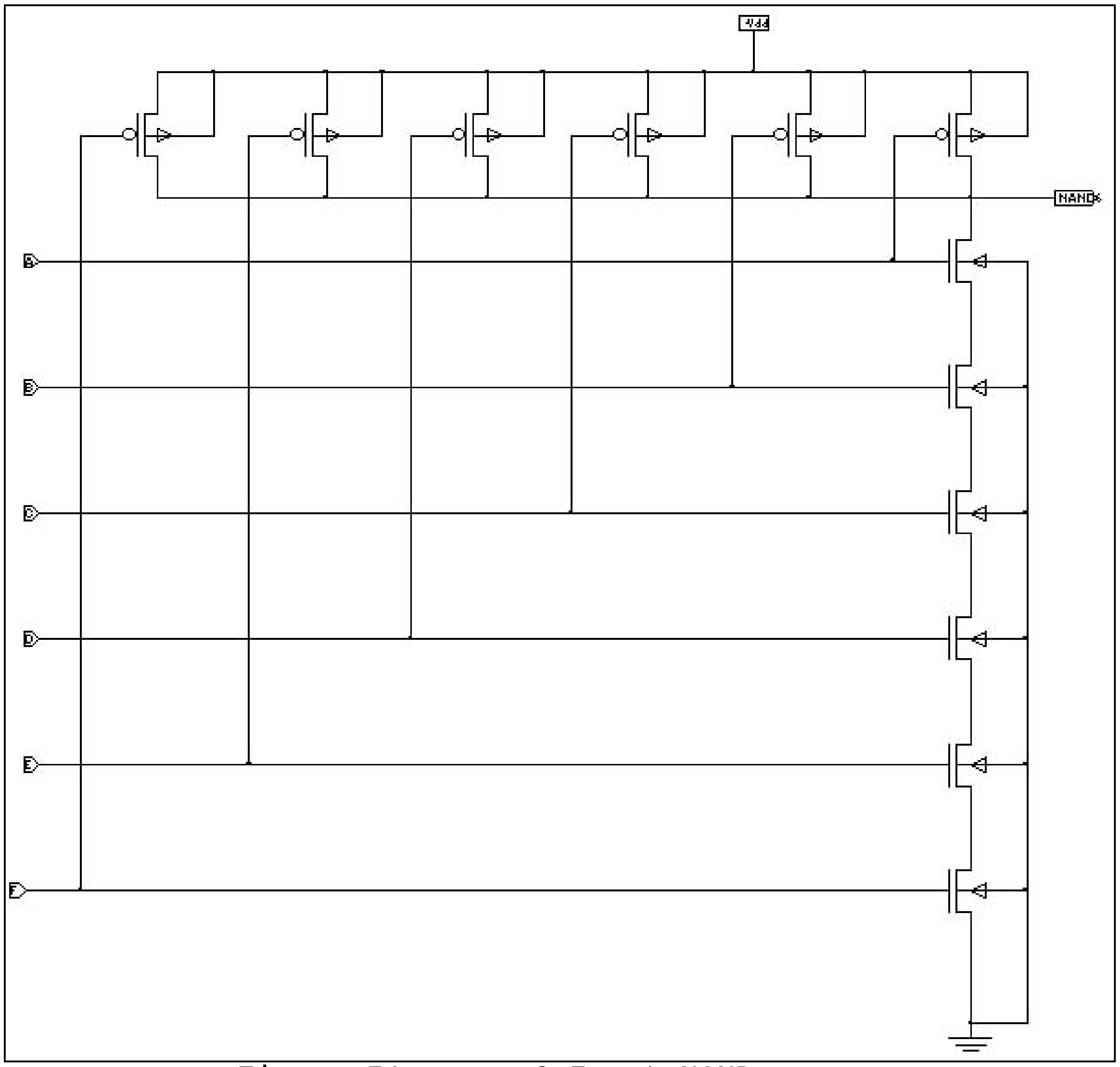


Figure 51. 6-Input NAND.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 15, 2001 at 09:50:35

* Main circuit: caa_NAND6
M1 NAND6 A N6 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N6 B N5 Gnd CMOSN W=5*lambda L=2*lambda
```

```
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C N4 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N4 D N2 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N2 E N1 Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 N1 F Gnd Gnd CMOSN W=5*lambda L=2*lambda
     AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
     PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
     PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M7 NAND6 E Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M8 NAND6 D Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M9 NAND6 C Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M10 NAND6 B Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M11 NAND6 A Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M12 NAND6 F Vdd Vdd CMOSP W=8*lambda L=2*lambda
     AS=8*lambda*5.5*lambda AD=8*lambda*5.5*lambda
     PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
     PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND6
.END
```

f. 8-Input

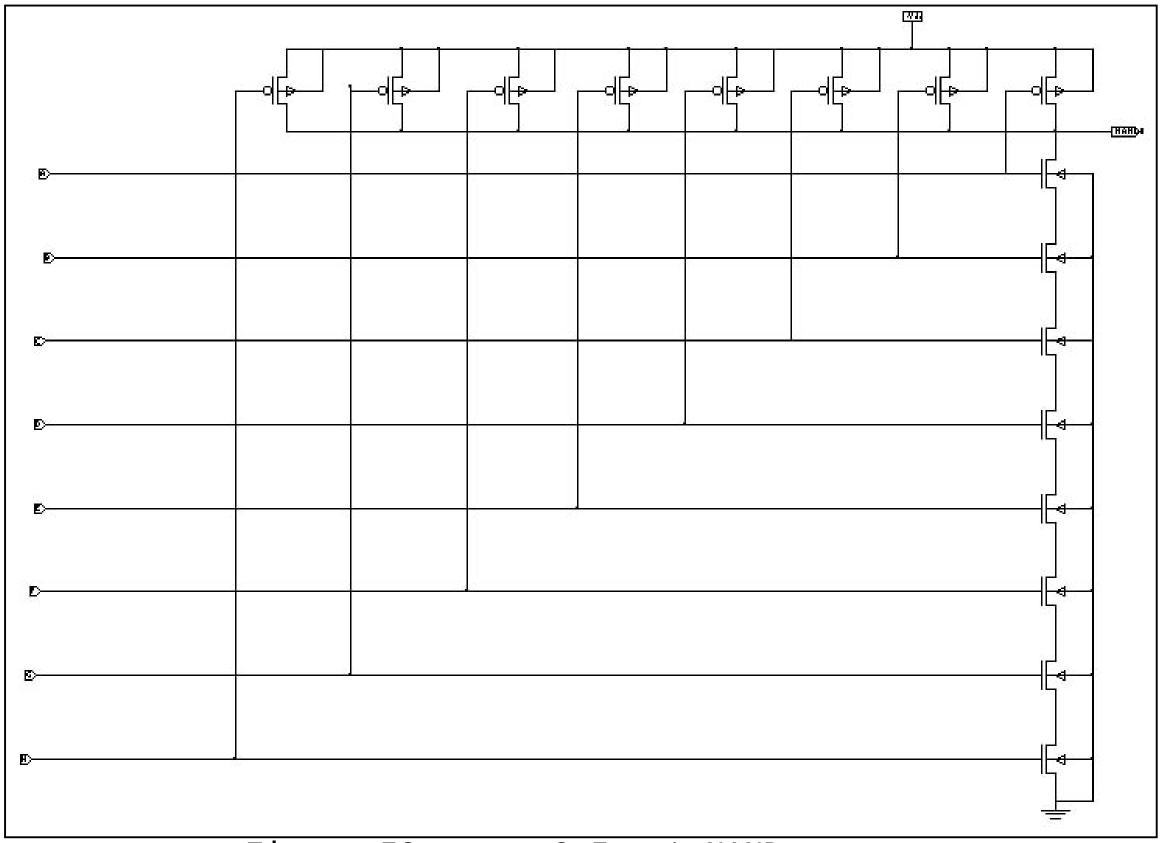


Figure 52. 8-Input NAND.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Aug 24, 2001 at 15:09:10

* Main circuit: caa_NAND8
M1 NAND8 A N1 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B N5 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C N8 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N8 D N11 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N11 E N14 Gnd CMOSN W=5*lambda L=2*lambda
```

```
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 N14 F N17 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M7 N17 G N20 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M8 N20 H Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M9 NAND8 G Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M10 NAND8 H Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M11 NAND8 F Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M12 NAND8 E Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M13 NAND8 D Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M14 NAND8 C Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M15 NAND8 B Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M16 NAND8 A Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND8
.END
```

g. 9-Input

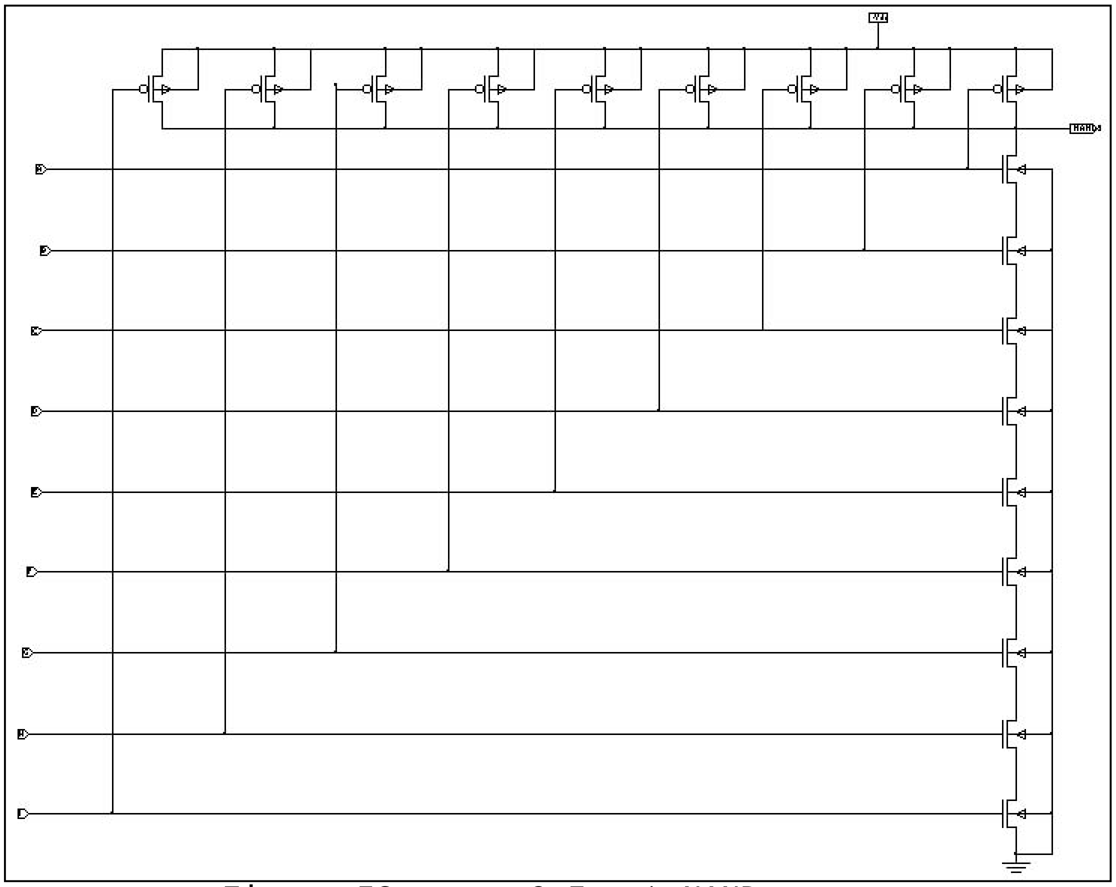


Figure 53. 9-Input NAND.

```
* SPICE netlist written by S-Edit Win32 6.00
* Main circuit: caa_NAND9
M1 NAND9 A N1 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B N5 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C N8 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N8 D N11 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N11 E N14 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 N14 F N10 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M7 N10 G N12 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M8 N12 H N15 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M9 N15 I Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M10 NAND9 G Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M11 NAND9 H Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M12 NAND9 I Vdd N25 CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M13 NAND9 F Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M14 NAND9 E Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M15 NAND9 D Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M16 NAND9 C Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M17 NAND9 B Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M18 NAND9 A Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND9
.END
```

h. 13-Input

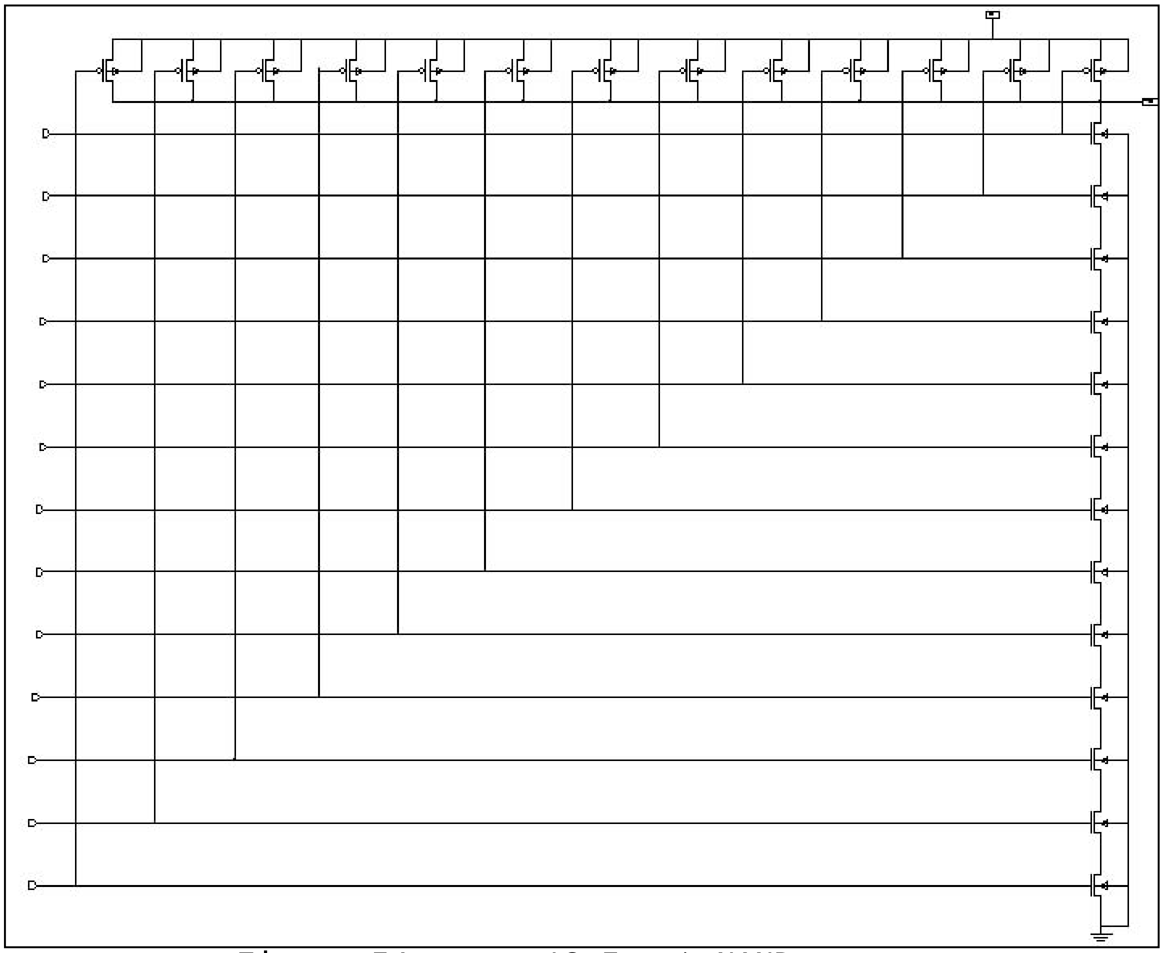


Figure 54. 13-Input NAND.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 15, 2001 at 10:34:36

* Main circuit: caa_NAND13
M1 N13 J N15 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M2 N15 K N18 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M3 N18 L N1 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M4 NAND13 A N4 Gnd CMOSN W=10*lambda L=2*lambda
```

```
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M5 N4 B N6 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M6 N6 C N8 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M7 N8 D N14 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M8 N14 E N9 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M9 N9 F N17 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M10 N17 G N23 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M11 N23 H N12 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M12 N12 I N13 Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M13 N1 M Gnd Gnd CMOSN W=10*lambda L=2*lambda
      AS=5.5*lambda*10*lambda AD=5.5*lambda*10*lambda
      PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
      PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M14 NAND13 I Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M15 NAND13 H Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M16 NAND13 G Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M17 NAND13 F Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
```

```
M18 NAND13 E Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M19 NAND13 D Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M20 NAND13 C Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M21 NAND13 B Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M22 NAND13 A Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M23 NAND13 J Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M24 NAND13 K Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M25 NAND13 L Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M26 NAND13 M Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND13
.END
```

i. 15-Input

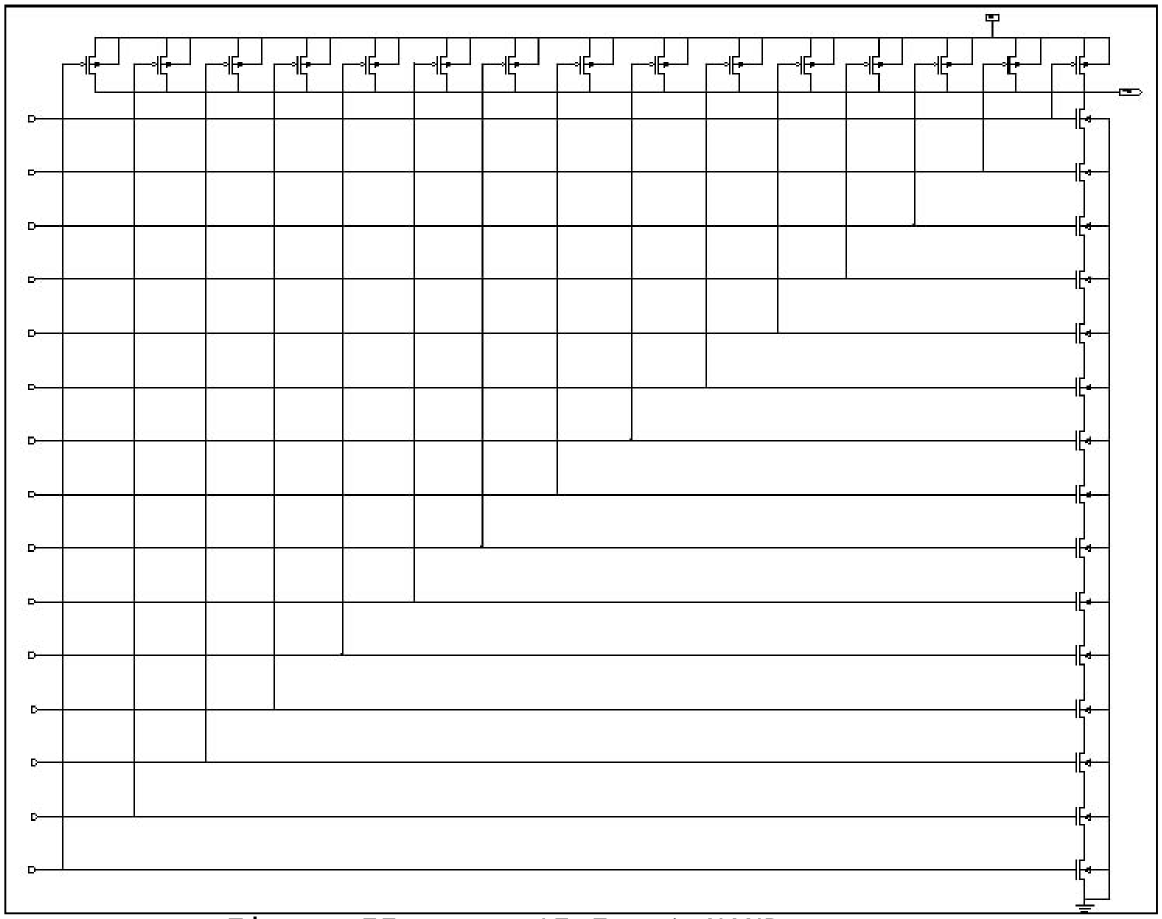


Figure 55. 15-Input NAND.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 20, 2001 at 21:20:16

* Main circuit: caa_NAND15
M1 N6 N N4 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M2 N4 O Gnd Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M3 N5 M N6 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M4 N7 L N5 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
```

```
M5 N2 K N7 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M6 N8 J N2 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M7 N3 I N8 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M8 N9 H N3 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M9 N10 G N9 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M10 N12 F N10 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M11 N13 E N12 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M12 N14 D N13 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M13 N11 C N14 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M14 N15 B N11 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M15 NAND15 A N15 Gnd CMOSN W=15*lambda L=2*lambda
      AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
      PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
      PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M16 NAND15 I Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M17 NAND15 H Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M18 NAND15 G Vdd Vdd CMOSP W=5*lambda L=2*lambda
```

```
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M19 NAND15 F Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M20 NAND15 E Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M21 NAND15 D Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M22 NAND15 C Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M23 NAND15 B Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M24 NAND15 A Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M25 NAND15 J Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M26 NAND15 K Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M27 NAND15 L Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M28 NAND15 M Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M29 NAND15 N Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M30 NAND15 O Vdd Vdd CMOSP W=5*lambda L=2*lambda
      AS=5*lambda*5.5*lambda AD=5*lambda*5.5*lambda
      PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NAND15
.END
```

3. NOR

a. 2-Input

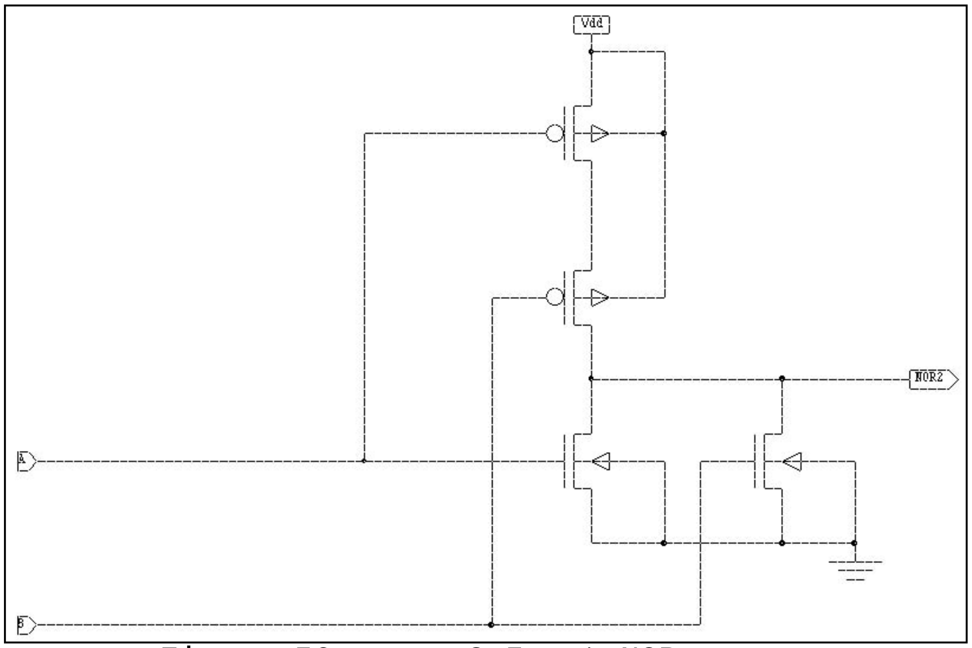


Figure 56. 2-Input NOR.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 29, 2001 at 08:39:50

* Main circuit: caa_NOR2
M1 NOR2 A Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 NOR2 B Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 NOR2 B N5 Vdd CMOSP W=17*lambda L=2*lambda
      AS=17*lambda*5.5*lambda AD=17*lambda*5.5*lambda
      PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
      PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
M4 N5 A Vdd Vdd CMOSP W=17*lambda L=2*lambda
      AS=17*lambda*5.5*lambda AD=17*lambda*5.5*lambda
      PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
      PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NOR2
.END
```

b. 4-Input

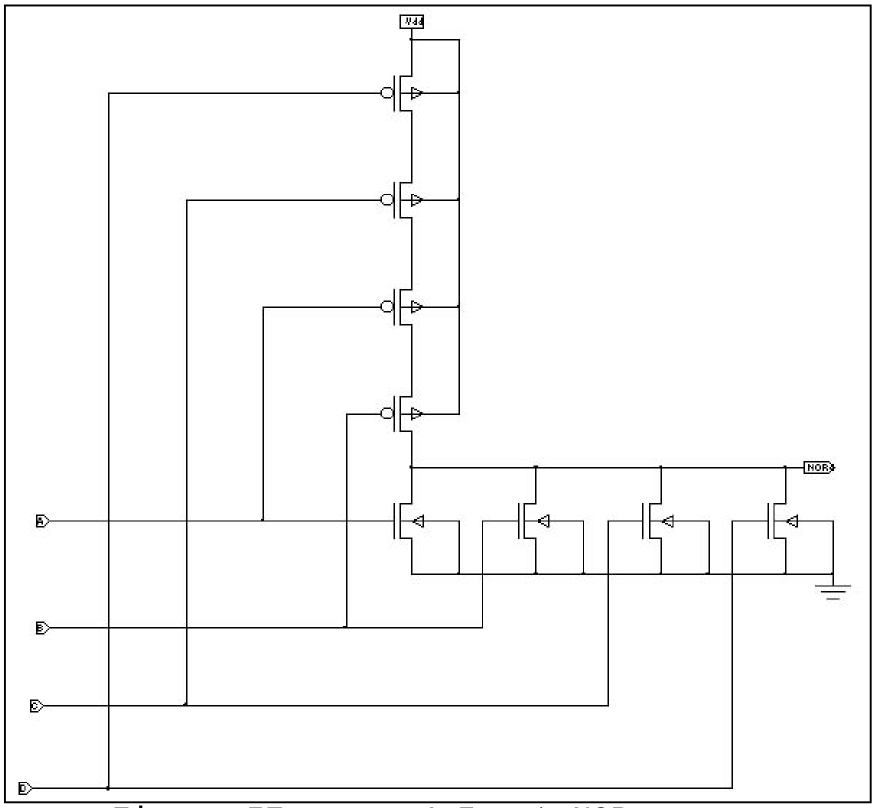


Figure 57. 4-Input NOR.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 20, 2001 at 20:46:37
* Main circuit: caa_NOR4
M1 NOR4 A Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 NOR4 B Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 NOR4 C Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 NOR4 D Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 NOR4 B N5 Vdd CMOSP W=17*lambda L=2*lambda
      AS=17*lambda*5.5*lambda AD=17*lambda*5.5*lambda
      PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
      PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
```

```
M6 N5 A N1 Vdd CMOSP W=17*lambda L=2*lambda
      AS=17*lambda*5.5*lambda AD=17*lambda*5.5*lambda
      PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
      PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
M7 N1 C N3 Vdd CMOSP W=17*lambda L=2*lambda
      AS=17*lambda*5.5*lambda AD=17*lambda*5.5*lambda
      PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
      PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
M8 N3 D Vdd Vdd CMOSP W=17*lambda L=2*lambda
      AS=17*lambda*5.5*lambda AD=17*lambda*5.5*lambda
      PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
      PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_NOR4
.END
```

4. 2X1 Multiplexer

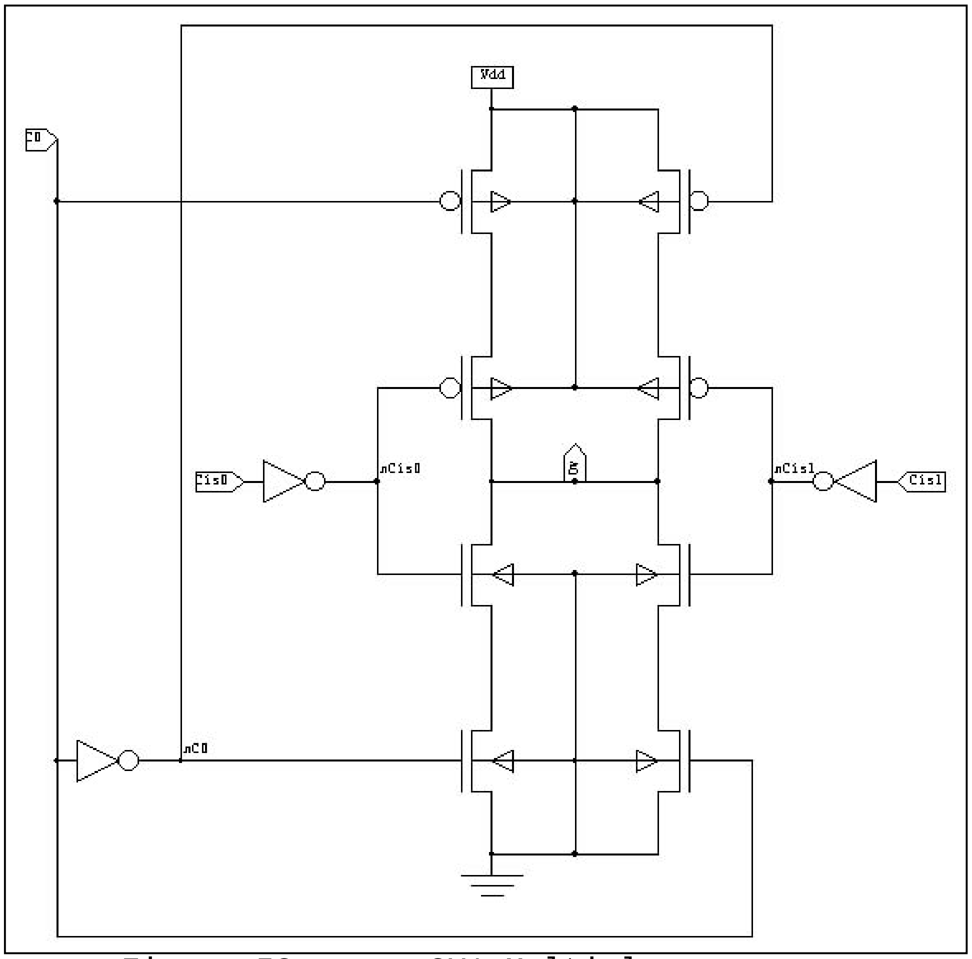


Figure 58. 2X1 Multiplexer.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Aug 24, 2001 at 15:59:26

* Main circuit: caa_MUX_2x1_ACT
Xcaa_INV_1 C0 nC0 Gnd Vdd caa_INV
Xcaa_INV_2 Cis0 nCis0 Gnd Vdd caa_INV
Xcaa_INV_3 Cis1 nCis1 Gnd Vdd caa_INV
M1 N4 nC0 Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 Ov nCis0 N4 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N10 C0 Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 Ov nCis1 N10 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N3 C0 Vdd Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M6 Ov nCis0 N3 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M7 N8 nC0 Vdd Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M8 Ov nCis1 N8 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_MUX_2x1_ACT
.END
```

C. LEVEL THREE – SECONDARY SUB-CIRCUITS

1. 4-Bit ALU Group Generate and Propagate (Grp GP)

a. Logic Group GP

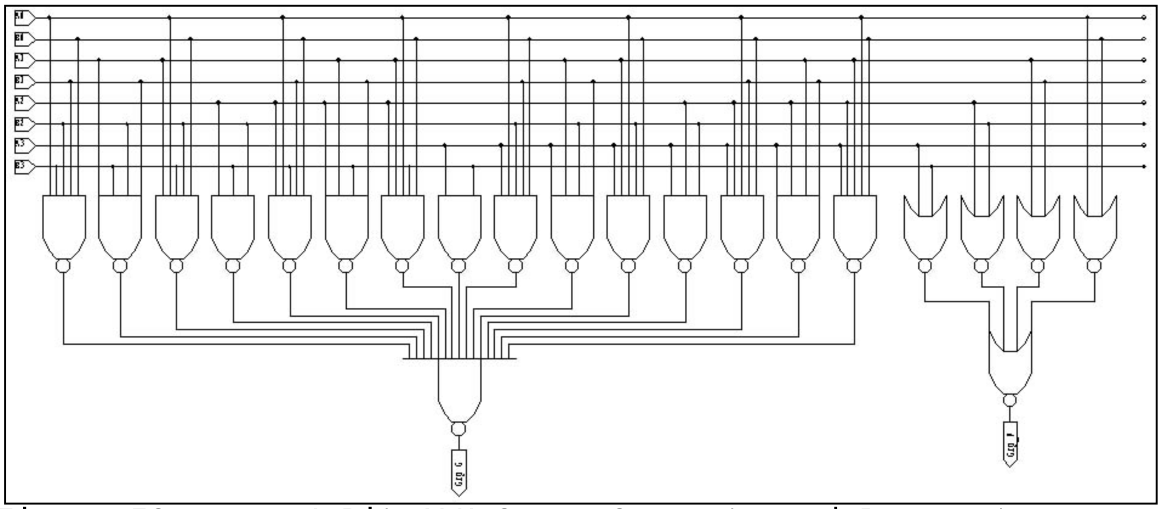


Figure 59. 4-Bit ALU Group Generate and Propagate.

```
* Main circuit: caa_Grp_PG
Xcaa_NAND15_1 N75 N70 N64 N60 N54 N49 N43 N16 N34 N29 N23 N2 + N12 N7
Grp_G N1 Gnd Vdd caa_NAND15
Xcaa_NAND2_1 B3 A3 N16 Gnd Vdd caa_NAND2
Xcaa_NAND3_1 B2 A2 A3 N60 Gnd Vdd caa_NAND3
Xcaa_NAND3_2 B2 B3 A2 N2 Gnd Vdd caa_NAND3
Xcaa_NAND4_1 B1 B2 B3 A1 N7 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 B1 B3 A1 A2 N29 Gnd Vdd caa_NAND4
Xcaa_NAND4_4 B1 B2 A1 A3 N49 Gnd Vdd caa_NAND4
Xcaa_NAND4_5 B1 A1 A2 A3 N70 Gnd Vdd caa_NAND4
Xcaa_NAND5_1 B0 B1 B2 B3 A0 N1 Gnd Vdd caa_NAND5
Xcaa_NAND5_2 B0 B2 B3 A0 A1 N12 Gnd Vdd caa_NAND5
Xcaa_NAND5_3 B0 B1 B3 A0 A2 N23 Gnd Vdd caa_NAND5
Xcaa_NAND5_4 B0 B3 A0 A1 A2 N34 Gnd Vdd caa_NAND5
Xcaa_NAND5_5 B0 B1 B2 A0 A3 N43 Gnd Vdd caa_NAND5
Xcaa_NAND5_6 B0 B2 A0 A1 A3 N54 Gnd Vdd caa_NAND5
Xcaa_NAND5_7 B0 B1 A0 A2 A3 N64 Gnd Vdd caa_NAND5
Xcaa_NAND5_8 B0 A0 A1 A2 A3 N75 Gnd Vdd caa_NAND5
Xcaa_NOR2_1 B3 A3 N3 Gnd Vdd caa_NOR2
Xcaa_NOR2_2 B2 A2 N5 Gnd Vdd caa_NOR2
Xcaa_NOR2_3 B1 A1 N10 Gnd Vdd caa_NOR2
Xcaa_NOR2_4 B0 A0 N14 Gnd Vdd caa_NOR2
Xcaa_NOR4_1 N14 N10 N5 N3 Grp_P Gnd Vdd caa_NOR4
* End of main circuit: caa_Grp_PG
.END
```

b. Combinational Gate Group G

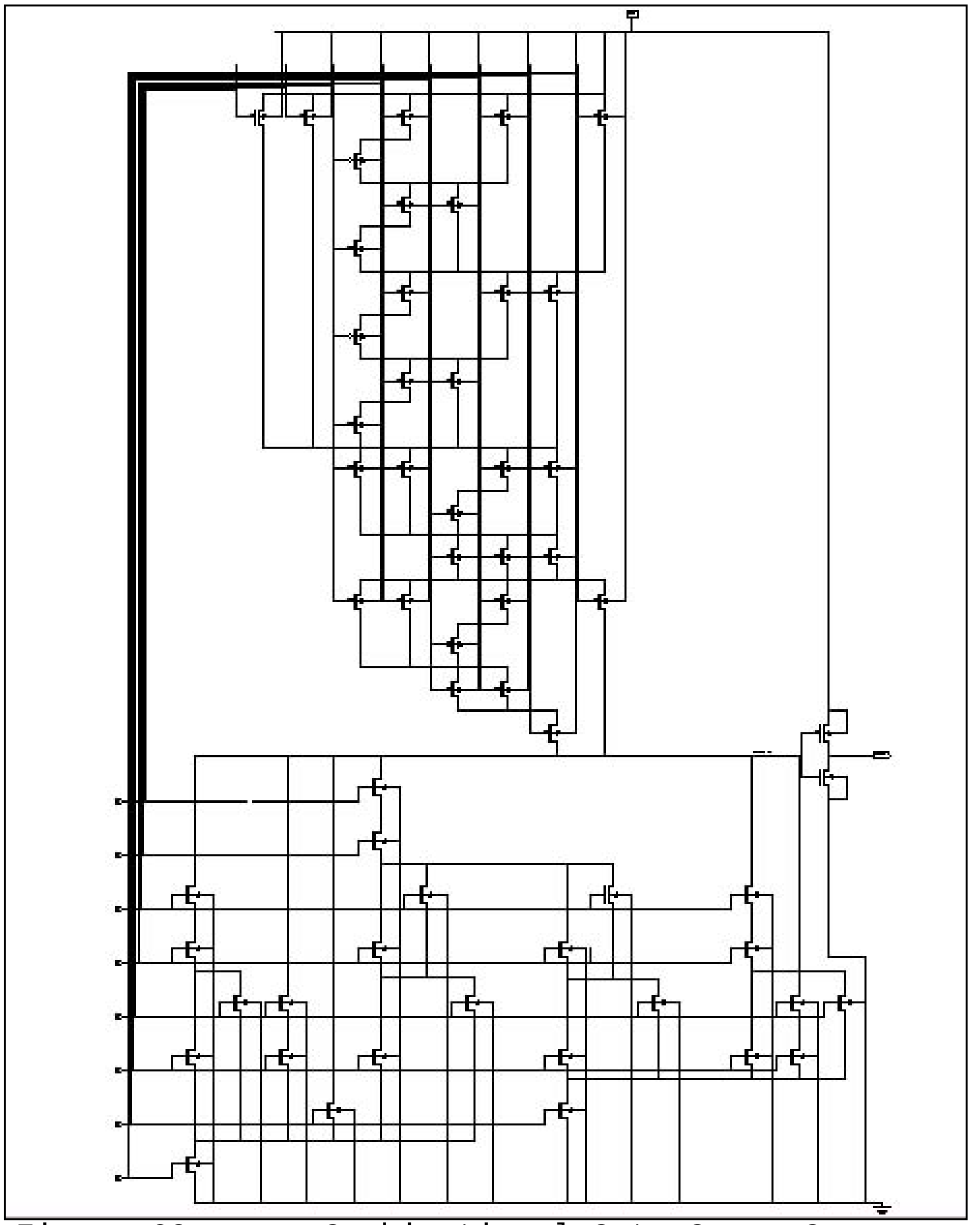


Figure 60. Combinational Gate Group G.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Aug 3, 2001 at 11:46:39

* Main circuit: caa_GrpG
M1 Gnd N1 Grp_G Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 A1 N10 Gnd CMOSN W=5*lambda L=2*lambda
```

```
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N9 A2 N110 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N8 B2 N110 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N1 A2 N8 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 N1 A3 N110 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M7 N6 B2 N110 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M8 N7 B1 N6 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M9 N114 B0 N7 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M10 N1 A0 N114 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M11 N7 A1 N6 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M12 N6 A2 N110 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M13 N106 A3 Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M14 N4 B2 N106 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M15 N7 B1 N4 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
```

```
M16 N10 B1 N9 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M17 N7 A1 N4 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M18 N9 B2 N110 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M19 N4 A2 N106 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M20 N2 B2 N106 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M21 N117 B1 N2 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M22 N1 A1 N117 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M23 N118 B2 N106 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M24 N1 A2 N118 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M25 N110 B3 Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M26 N2 A2 N106 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M27 N16 A0 Vdd Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M28 N16 B0 Vdd Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M29 N17 A1 N19 Vdd CMOSP W=14*lambda L=2*lambda
```

```
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M30 N19 B1 Vdd Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M31 N20 B1 N17 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M32 N21 A1 N20 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M33 N21 A2 N17 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M34 N17 B2 Vdd Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M35 N21 B3 Vdd Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M36 N16 A3 N21 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M37 N23 B2 N21 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M38 N22 B1 N21 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M39 N23 A1 N22 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M40 N16 A2 N23 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M41 N24 B1 N23 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M42 N16 A1 N24 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
```

```
M43 N29 A1 N16 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M44 N29 B1 N16 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M45 N26 B2 N16 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M46 N29 A2 N26 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M47 N29 A3 N16 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M48 N25 A2 N29 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M49 N25 B2 N29 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M50 N25 A3 N29 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M51 N30 A1 N25 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M52 N30 B1 N25 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M53 N27 B2 N25 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M54 N30 A2 N27 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M55 N1 B3 N25 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M56 N28 A2 N30 Vdd CMOSP W=14*lambda L=2*lambda
```

```
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M57 N28 B2 N30 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M58 N1 A3 N28 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M59 Vdd N1 Grp_G Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_GrpG
.END
```

c. Combinational Gate Group P

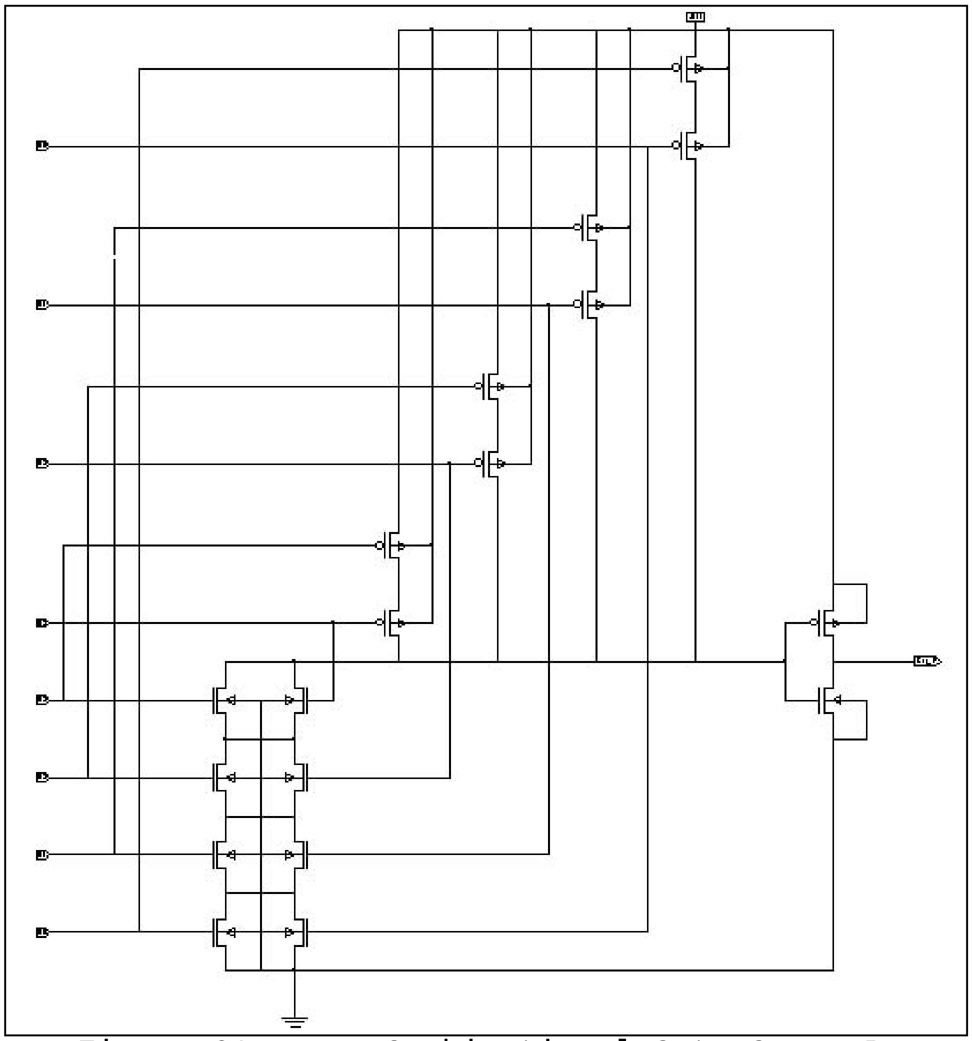


Figure 61. Combinational Gate Group P.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Jul 24, 2001 at 17:03:52

* Main circuit: caa_GrpP
M1 Gnd A0 N14 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N14 B0 Gnd Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N13 B1 N14 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N14 A1 N13 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N12 B2 N13 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 N13 A2 N12 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M7 N8 B3 N12 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M8 N12 A3 N8 Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M9 Gnd N8 Grp_P Gnd CMOSN W=5*lambda L=2*lambda
      AS=5.5*lambda*5*lambda AD=5.5*lambda*5*lambda
      PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
      PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M10 N4 B3 N8 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M11 Vdd A3 N4 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M12 N3 B2 N8 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M13 Vdd A2 N3 Vdd CMOSP W=14*lambda L=2*lambda
```

```
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M14 N2 B1 N8 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M15 Vdd A1 N2 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M16 N1 B0 N8 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M17 Vdd A0 N1 Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M18 Vdd N8 Grp_P Vdd CMOSP W=14*lambda L=2*lambda
      AS=14*lambda*5.5*lambda AD=14*lambda*5.5*lambda
      PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
      PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
* End of main circuit: caa_GrpP
.END
```

2. 4-Bit ALU Generate and Propagate (GP)

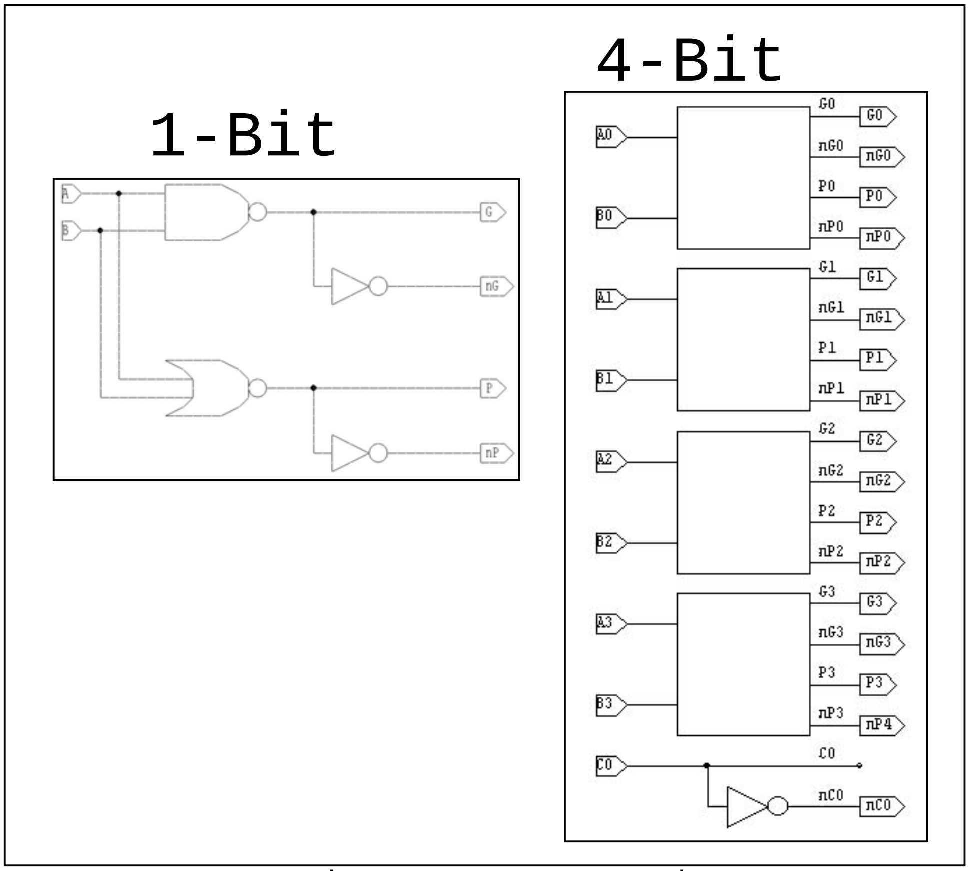


Figure 62. 4-Bit ALU Generate and Propagate.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 15, 2001 at 13:34:17

.SUBCKT caa_PG_GENx1 A B G nG nP P Gnd Vdd
Xcaa_INV_1 G nG Gnd Vdd caa_INV
Xcaa_INV_2 P nP Gnd Vdd caa_INV
Xcaa_NAND2_1 A B G Gnd Vdd caa_NAND2
Xcaa_NOR2_1 A B P Gnd Vdd caa_NOR2
.ENDS

* Main circuit: PG_GENx4
Xcaa_PG_GENx1_1 A0 B0 G0 nG0 nP0 P0 Gnd Vdd caa_PG_GENx1
Xcaa_PG_GENx1_2 A1 B1 G1 nG1 nP1 P1 Gnd Vdd caa_PG_GENx1
Xcaa_PG_GENx1_3 A2 B2 G2 nG2 nP2 P2 Gnd Vdd caa_PG_GENx1
Xcaa_PG_GENx1_4 A3 B3 G3 nG3 nP4 P3 Gnd Vdd caa_PG_GENx1
* End of main circuit: PG_GENx4
.END
```

3. 4-Bit Overflow

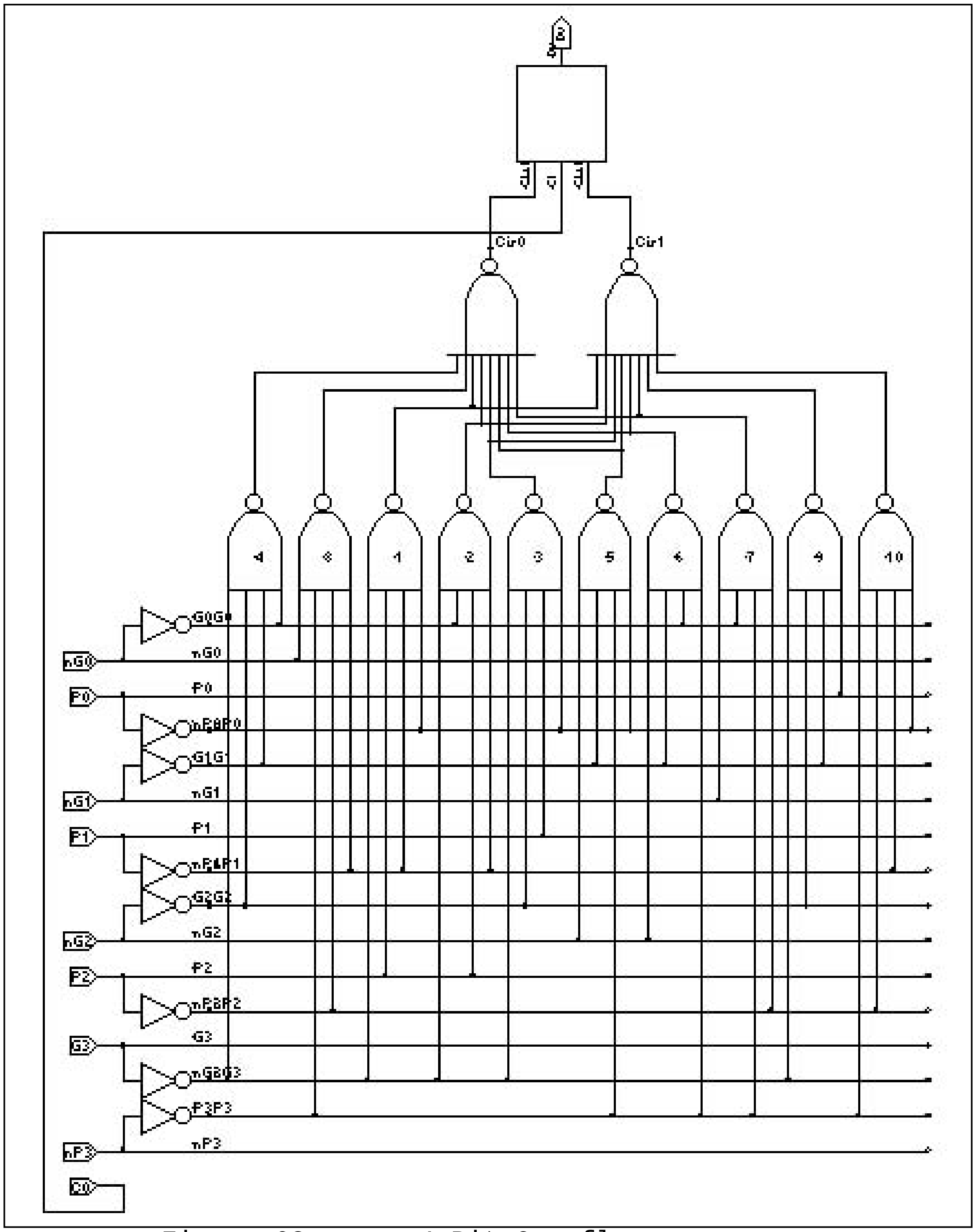


Figure 63. 4-Bit Overflow.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Aug 23, 2001 at 14:03:42
* Main circuit: caa_overflow
Xcaa_INV_1 nG0 G0 Gnd Vdd caa_INV
Xcaa_INV_2 P0 nP0 Gnd Vdd caa_INV
Xcaa_INV_3 nG1 G1 Gnd Vdd caa_INV
Xcaa_INV_4 P1 nP1 Gnd Vdd caa_INV
Xcaa_INV_5 nG2 G2 Gnd Vdd caa_INV
Xcaa_INV_6 P2 nP2 Gnd Vdd caa_INV
Xcaa_INV_7 G3 nG3 Gnd Vdd caa_INV
Xcaa_INV_8 nP3 P3 Gnd Vdd caa_INV
Xcaa_MUX_2x1_ACT_1 C0 N3 N2 Ov Gnd Vdd caa_MUX_2x1_ACT
Xcaa_NAND4_1 nG3 G2 G1 G0 N1 Gnd Vdd caa_NAND4
Xcaa_NAND4_2 nG0 P3 nP2 nP1 N10 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 nG3 P2 nP1 nP0 N9 Gnd Vdd caa_NAND4
Xcaa_NAND4_4 nG3 G0 P2 nP1 N8 Gnd Vdd caa_NAND4
Xcaa_NAND4_5 nG3 G2 P1 nP0 N7 Gnd Vdd caa_NAND4
Xcaa_NAND4_6 nG2 G1 P3 nP0 N19 Gnd Vdd caa_NAND4
Xcaa_NAND4_7 nG2 G1 G0 P3 N6 Gnd Vdd caa_NAND4
Xcaa_NAND4_8 nG1 G0 P3 nP2 N4 Gnd Vdd caa_NAND4
Xcaa_NAND4_9 nG3 G2 G1 P0 N5 Gnd Vdd caa_NAND4
Xcaa_NAND4_10 P3 nP2 nP1 nP0 N46 Gnd Vdd caa_NAND4
Xcaa_NAND8_1 N1 N10 N9 N8 N7 N19 N6 N17 N3 Gnd Vdd caa_NAND8
Xcaa_NAND8_2 N9 N8 N7 N19 N18 N4 N5 N46 N2 Gnd Vdd caa_NAND8
* End of main circuit: caa_overflow
.END
```

4. Carry Look-Ahead (CLAH)

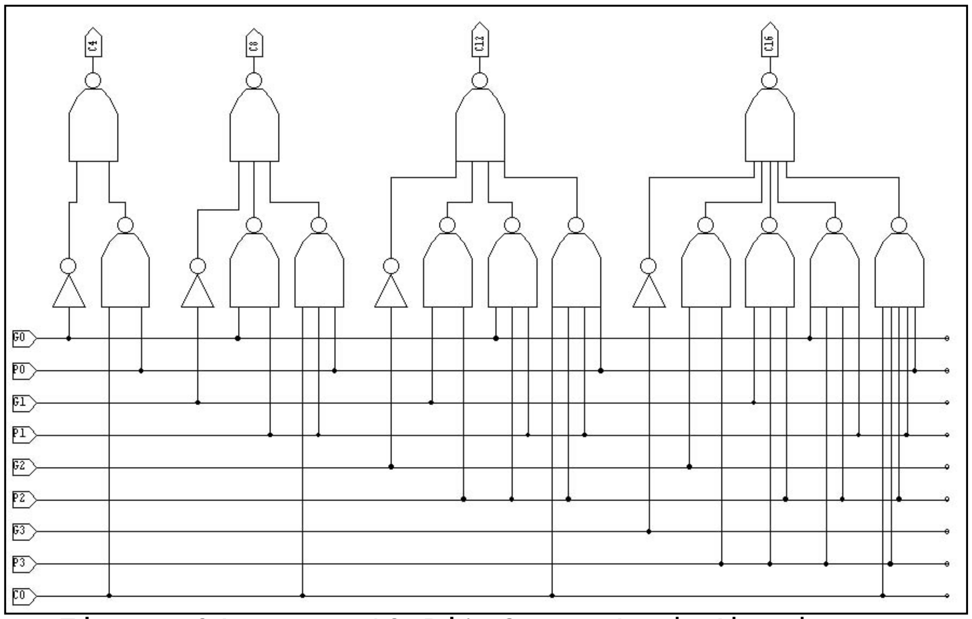


Figure 64. 16-Bit Carry Look-Ahead.

```
* SPICE netlist written by S-Edit Win32 6.00
* Written on May 25, 2001 at 10:48:45

* Main circuit: caa_CLAH
Xcaa_INV_1 G0 N28 Gnd Vdd caa_INV
Xcaa_INV_2 G1 N27 Gnd Vdd caa_INV
Xcaa_INV_3 G2 N26 Gnd Vdd caa_INV
Xcaa_INV_4 G3 N25 Gnd Vdd caa_INV
Xcaa_NAND2_1 C0 P0 N24 Gnd Vdd caa_NAND2
Xcaa_NAND2_2 G0 P1 N23 Gnd Vdd caa_NAND2
Xcaa_NAND2_3 G1 P2 N22 Gnd Vdd caa_NAND2
Xcaa_NAND2_4 G2 P3 N21 Gnd Vdd caa_NAND2
Xcaa_NAND2_5 N28 N24 C4 Gnd Vdd caa_NAND2
Xcaa_NAND3_1 C0 P1 P0 N18 Gnd Vdd caa_NAND3
Xcaa_NAND3_2 G0 P2 P1 N17 Gnd Vdd caa_NAND3
Xcaa_NAND3_3 G1 P3 P2 N16 Gnd Vdd caa_NAND3
Xcaa_NAND3_4 N27 N23 N18 C8 Gnd Vdd caa_NAND3
Xcaa_NAND4_1 C0 P2 P1 P0 N12 Gnd Vdd caa_NAND4
Xcaa_NAND4_2 G0 P3 P2 P1 N11 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 N26 N22 N17 N12 C12 Gnd Vdd caa_NAND4
Xcaa_NAND5_1 C0 P3 P2 P1 P0 N6 Gnd Vdd caa_NAND5
Xcaa_NAND5_2 N25 N21 N16 N11 N6 C16 Gnd Vdd caa_NAND5
* End of main circuit: caa_CLAH
.END
```

D. LEVEL FOUR – 4-BIT ALU

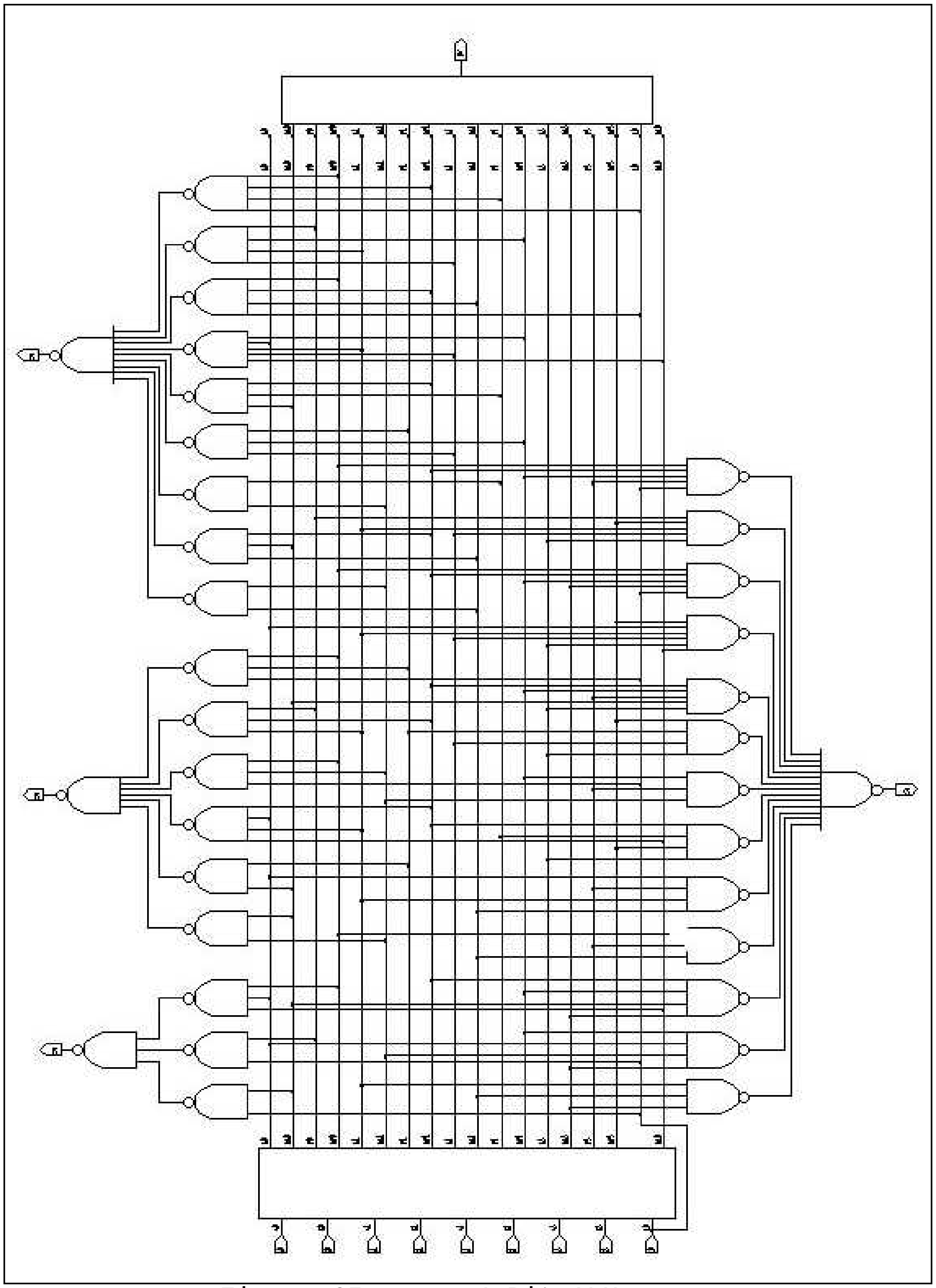


Figure 65. 4-Bit ALU.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Aug 23, 2001 at 14:04:28
* Main circuit: caa_Adderx4
Xcaa_NAND13_1 N45 N44 N43 N42 N40 N38 N37 N39 N41 N46 N47
+ N48 N49 S3 Gnd Vdd caa_NAND13
Xcaa_NAND2_1 C0 nG0 N20 Gnd Vdd caa_NAND2
Xcaa_NAND2_2 C0 P0 N19 Gnd Vdd caa_NAND2
Xcaa_NAND2_3 nG1 nG0 N27 Gnd Vdd caa_NAND2
Xcaa_NAND2_4 nG0 P1 N26 Gnd Vdd caa_NAND2
Xcaa_NAND2_5 nG2 nG1 N34 Gnd Vdd caa_NAND2
Xcaa_NAND2_6 nG1 P2 N32 Gnd Vdd caa_NAND2
Xcaa_NAND3_1 nC0 G0 nP0 N21 Gnd Vdd caa_NAND3
Xcaa_NAND3_2 C0 nG1 nP0 N23 Gnd Vdd caa_NAND3
Xcaa_NAND3_3 G1 nP1 P0 N24 Gnd Vdd caa_NAND3
Xcaa_NAND3_4 C0 P1 nP0 N25 Gnd Vdd caa_NAND3
Xcaa_NAND3_5 nG2 nG0 nP1 N33 Gnd Vdd caa_NAND3
Xcaa_NAND3_6 G2 nP2 P1 N31 Gnd Vdd caa_NAND3
Xcaa_NAND3_7 G1 nG2 nG3 N49 Gnd Vdd caa_NAND3
Xcaa_NAND3_8 nP0 P3 nG2 N46 Gnd Vdd caa_NAND3
Xcaa_NAND3_9 nP2 P3 nG1 N37 Gnd Vdd caa_NAND3
Xcaa_NAND3_10 nG0 P2 nP1 N29 Gnd Vdd caa_NAND3
Xcaa_NAND3_11 N20 N19 N21 S0 Gnd Vdd caa_NAND3
Xcaa_NAND4_1 nC0 G1 G0 nP1 N22 Gnd Vdd caa_NAND4
Xcaa_NAND4_2 C0 nG2 nP1 nP0 N30 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 G2 G1 nP2 P0 N35 Gnd Vdd caa_NAND4
Xcaa_NAND4_4 C0 P2 nP1 nP0 N36 Gnd Vdd caa_NAND4
Xcaa_NAND4_5 nP2 G0 nG1 nG3 N48 Gnd Vdd caa_NAND4
Xcaa_NAND4_6 nP1 nP2 nG0 nG3 N47 Gnd Vdd caa_NAND4
Xcaa_NAND4_7 G0 P3 G1 nG2 N41 Gnd Vdd caa_NAND4
Xcaa_NAND4_8 nP1 P2 nP3 G3 N39 Gnd Vdd caa_NAND4
Xcaa_NAND4_9 nP3 P1 G2 G3 N38 Gnd Vdd caa_NAND4
Xcaa_NAND5_1 nC0 G2 G1 G0 nP2 N28 Gnd Vdd caa_NAND5
Xcaa_NAND5_2 nP1 nP2 P3 nG0 G3 N40 Gnd Vdd caa_NAND5
Xcaa_NAND5_3 nP0 nP1 nP2 nG3 C0 N43 Gnd Vdd caa_NAND5
Xcaa_NAND5_4 P0 nP3 G1 G2 G3 N44 Gnd Vdd caa_NAND5
Xcaa_NAND5_5 nP0 nP1 nP2 P3 C0 N45 Gnd Vdd caa_NAND5
Xcaa_NAND6_1 nP3 G0 G1 G2 G3 nC0 N42 Gnd Vdd caa_NAND6
Xcaa_NAND6_2 N27 N26 N22 N23 N24 N25 S1 Gnd Vdd caa_NAND6
Xcaa_NAND9_1 N34 N33 N32 N31 N29 N28 N30 N35 N36 S2 Gnd Vdd caa_NAND9
Xcaa_overflow_1 C0 G3 nG0 nG1 nG2 nP3 Ov P0 P1 P2 Gnd Vdd
+ caa_overflow
XPG_GEN_1 A0 A1 A2 A3 B0 B1 B2 B3 C0 G0 G1 G2 G3 nC0 nG0 nG1
+ nG2 nG3 nP0 nP1 nP2 nP3 P0 P1 P2 P3 Gnd Vdd caa_PG_GENx4
* End of main circuit: caa_Adderx4
.END
```

E. LEVEL FIVE – 16-BIT ADDER

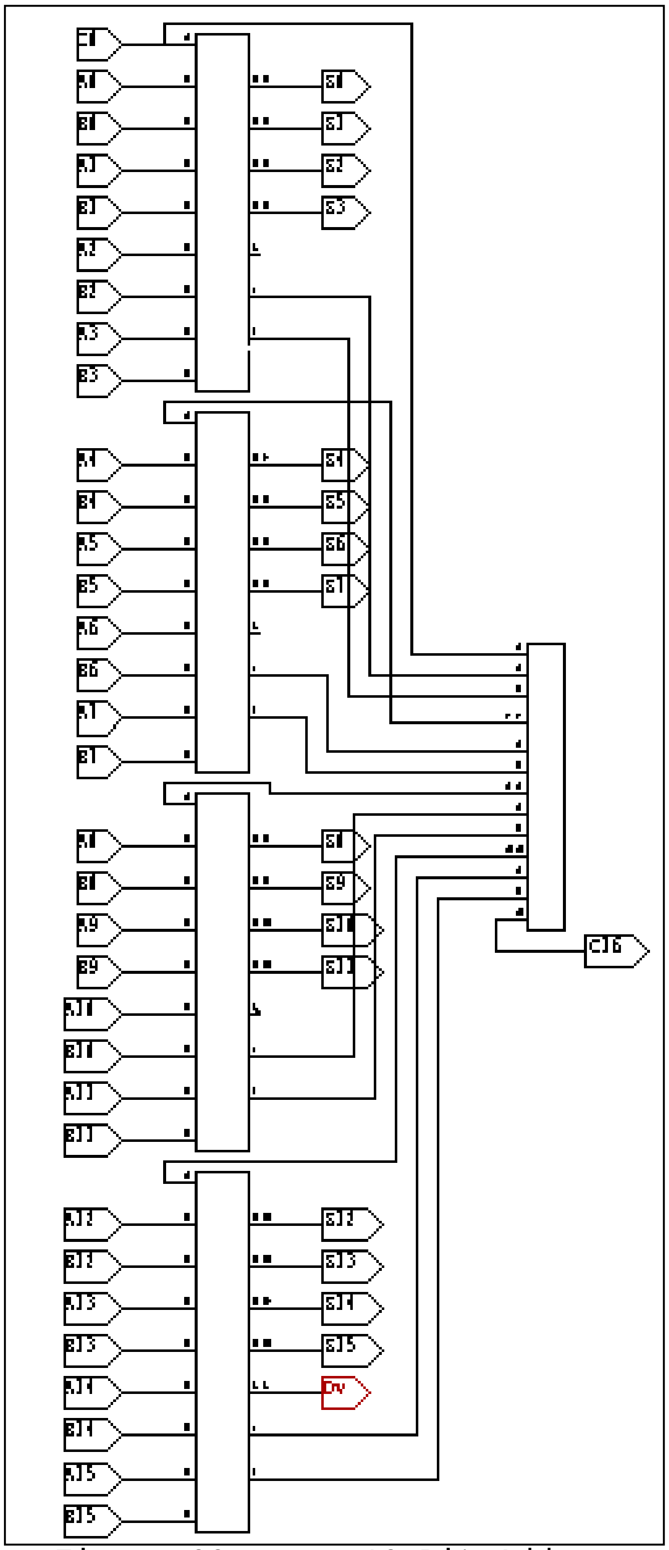


Figure 66. 16-Bit Adder.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Aug 23, 2001 at 19:37:19

* Main circuit: caa_Adderx16
XALUx4_1 A0 A1 A2 A3 B0 B1 B2 B3 C0 N11 N12 N1 S0 S1 S2 S3 Gnd Vdd ALUx4
XALUx4_2 A4 A5 A6 A7 B4 B5 B6 B7 N8 N9 N10 N2 S4 S5 S6 S7 Gnd Vdd ALUx4
XALUx4_3 A8 A9 A10 A11 B8 B9 B10 B11 N5 N6 N7 N3 S8 S9 S10 S11 Gnd Vdd
+ ALUx4
XALUx4_4 A12 A13 A14 A15 B12 B13 B14 B15 N14 N15 N4 Ov S12 S13 S14 S15
+ Gnd Vdd ALUx4
Xcaa_CLAH_1 C0 N8 N5 N14 C16 N11 N9 N6 N15 N12 N10 N7 N4 Gnd Vdd
+ caa_CLAH
* End of main circuit: caa_Adderx16
.END
```

APPENDIX B. FABRICATION LAYOUTS W/SPICE NET LISTS

A. LEVEL ONE – LOGIC GATES

1. Inverter

Due to the intricacies of placing gates in different sub-circuits, four inverters were utilized. They all perform the same logic function, but are slightly different in order to fit within a fabrication layout sub-circuit.

a. Inverter for CLAH

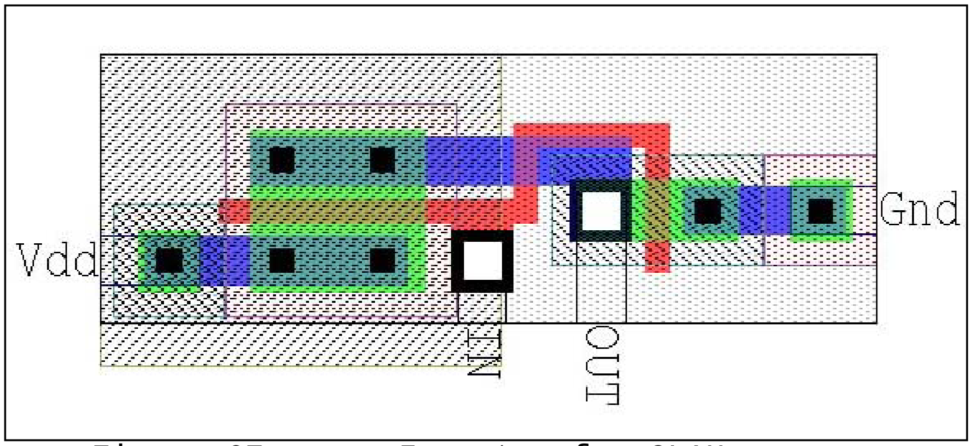


Figure 67. Inverter for CLAH.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23 * Cell:
caa_INV_CLAH      Version 1.39
* Extract Date and Time:  08/21/2001 - 19:54

.SUBCKT caa_INV_CLAH Gnd IN OUT Vdd

M1 OUT IN Gnd Gnd CMOSN L=180n W=450n
M2 Vdd IN OUT Vdd CMOSP L=180n W=1.26u

* Total Nodes: 4
* Total Elements: 2
.ENDS
```

b. *Inverter for Group GP*

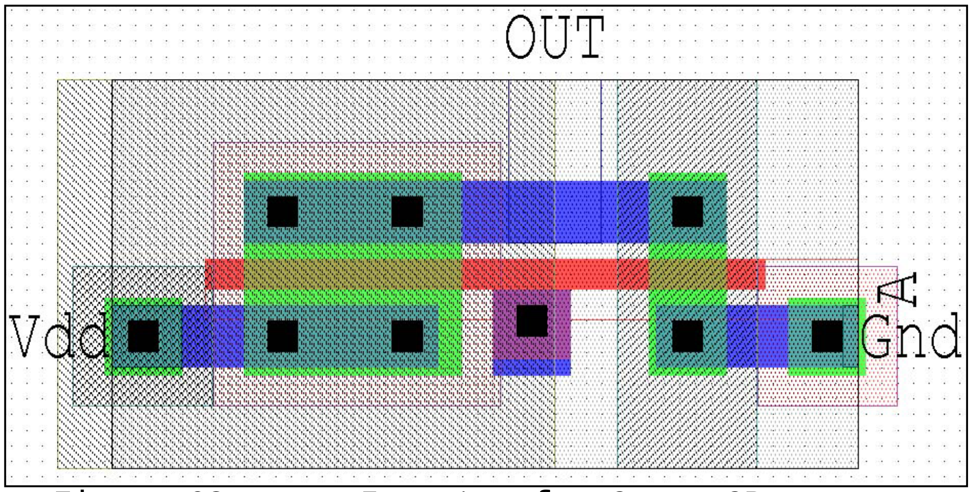


Figure 68. Inverter for Group GP.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_INV_Grp    Version 1.32
* Extract Date and Time:  08/06/2001 - 21:47

.SUBCKT caa_INV_Grp A Gnd OUT Vdd

M1 Gnd A OUT Gnd CMOSN L=180n W=450n
M2 Vdd A OUT Vdd CMOSP L=180n W=1.26u

* Total Nodes: 4
* Total Elements: 2
.ENDS
```

c. *Inverter for ALU GP*

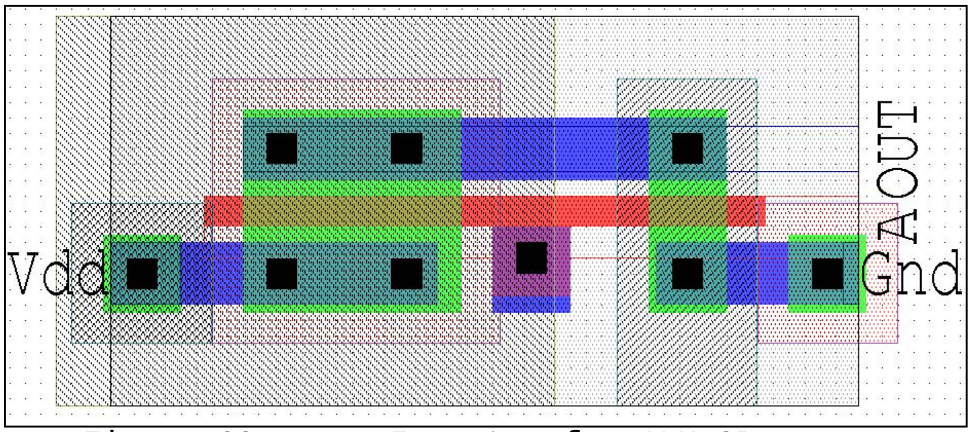


Figure 69. Inverter for ALU GP.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_INV_H      Version 1.31

.SUBCKT caa_INV_H A Gnd OUT Vdd

M1 Gnd A OUT Gnd CMOSN L=180n W=450n
M2 Vdd A OUT Vdd CMOSP L=180n W=1.26u

* Total Nodes: 4
* Total Elements: 2
.ENDS
```

d. Inverter for Overflow

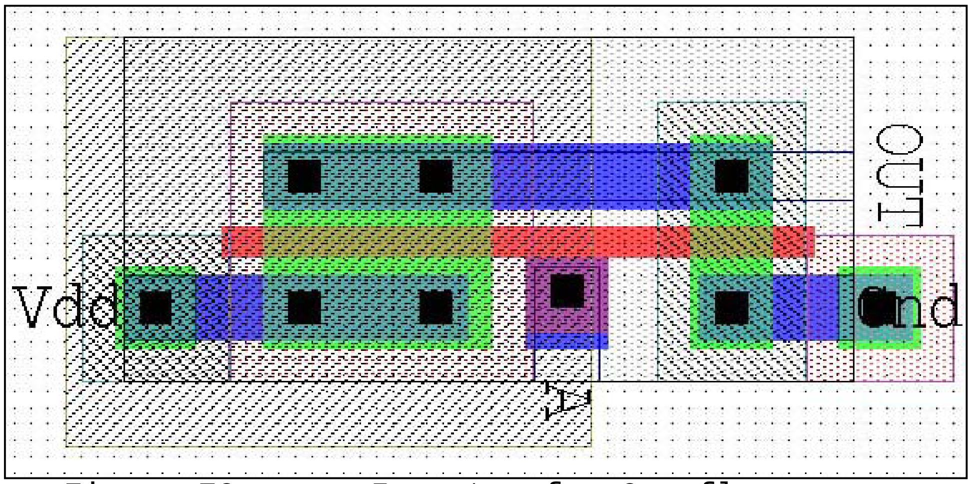


Figure 70. Inverter for Overflow.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_INV_Ov     Version 1.35

.SUBCKT caa_INV_Ov A Gnd OUT Vdd

M1 Gnd A OUT Gnd CMOSN L=180n W=450n
M2 Vdd A OUT Vdd CMOSP L=180n W=1.26u

* Total Nodes: 4
* Total Elements: 2
.ENDS
```

2. NAND

In a similar manner as discussed for the inverters, a number of versions of X-input NAND gates were used in different sub-circuits. They all perform the same logic function, but are slightly different in order to fit within the fabrication layout sub-circuit.

a. 2-Input

(1) 2-Input NAND

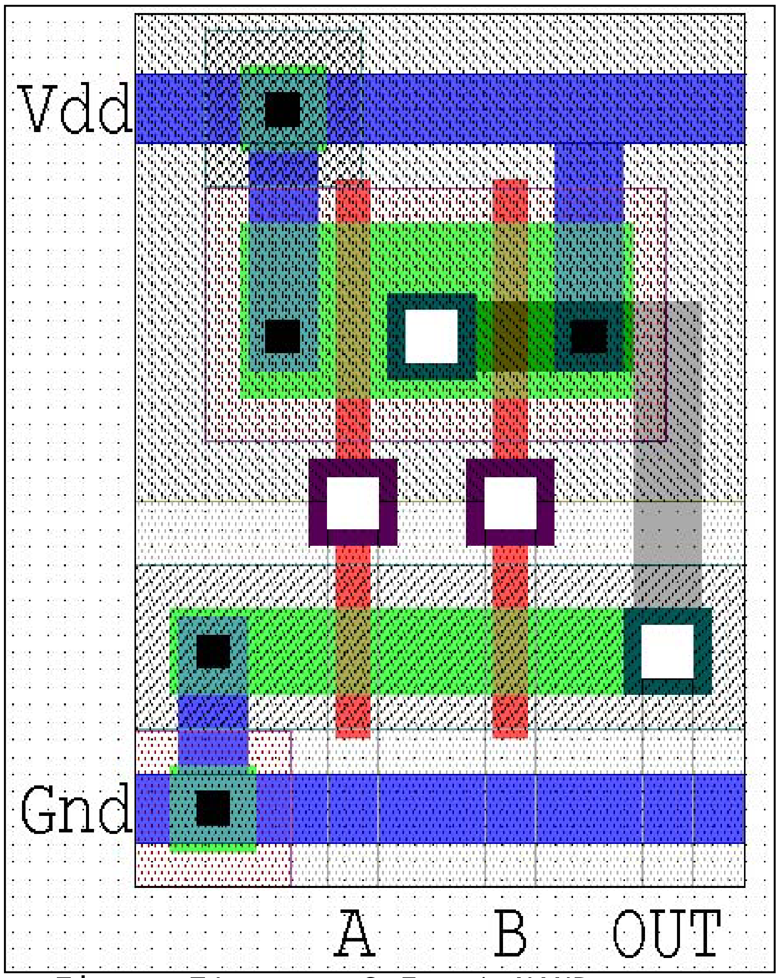


Figure 71. 2-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23 * Cell:
caa_NAND2   Version 1.24

.SUBCKT caa_NAND2 A B Gnd OUT Vdd

M1 OUT B 6 Gnd CMOSN L=180n W=450n
M2 6 A Gnd Gnd CMOSN L=180n W=450n
M3 Vdd B OUT Vdd CMOSP L=180n W=900n
M4 OUT A Vdd Vdd CMOSP L=180n W=900n

* Total Nodes: 6
* Total Elements: 4
.ENDS
```

(2) 2-Input NAND for ALU GP

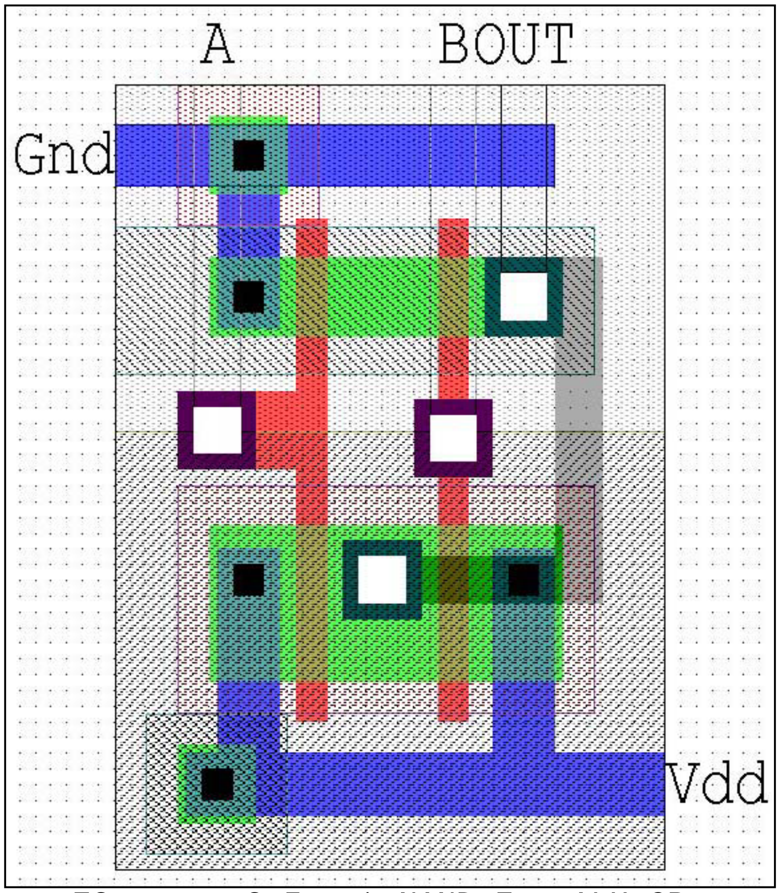


Figure 72. 2-Input NAND For ALU GP.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND2_H    Version 1.30
* Extract Date and Time:  07/25/2001 - 22:09

.SUBCKT caa_NAND2_H A B Gnd OUT Vdd

M1 6 B OUT Gnd CMOSN L=180n W=450n
M2 Gnd A 6 Gnd CMOSN L=180n W=450n
M3 OUT B Vdd Vdd CMOSP L=180n W=900n
M4 Vdd A OUT Vdd CMOSP L=180n W=900n

* Total Nodes: 6
* Total Elements: 4
.ENDS
```

(3) 2-Input NAND for Group GP

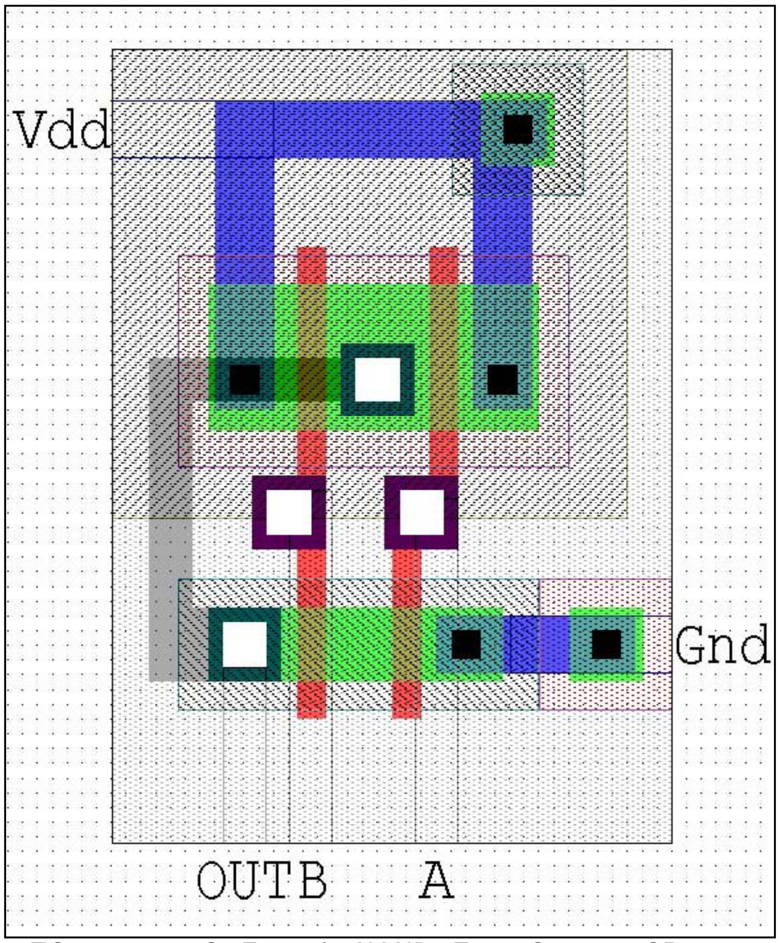


Figure 73. 2-Input NAND For Group GP.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23 * Cell:
caa_NAND2_GrpG    Version 1.37
* Extract Date and Time:  08/26/2001 - 15:23

.SUBCKT caa_NAND2_GrpG A B Gnd OUT Vdd

M1 OUT B 6 Gnd CMOSN L=180n W=450n
M2 6 A Gnd Gnd CMOSN L=180n W=450n
M3 Vdd B OUT Vdd CMOSP L=180n W=900n
M4 OUT A Vdd Vdd CMOSP L=180n W=900n

* Total Nodes: 6
* Total Elements: 4
.ENDS
```

b. 3-Input

(1) 3-Input NAND

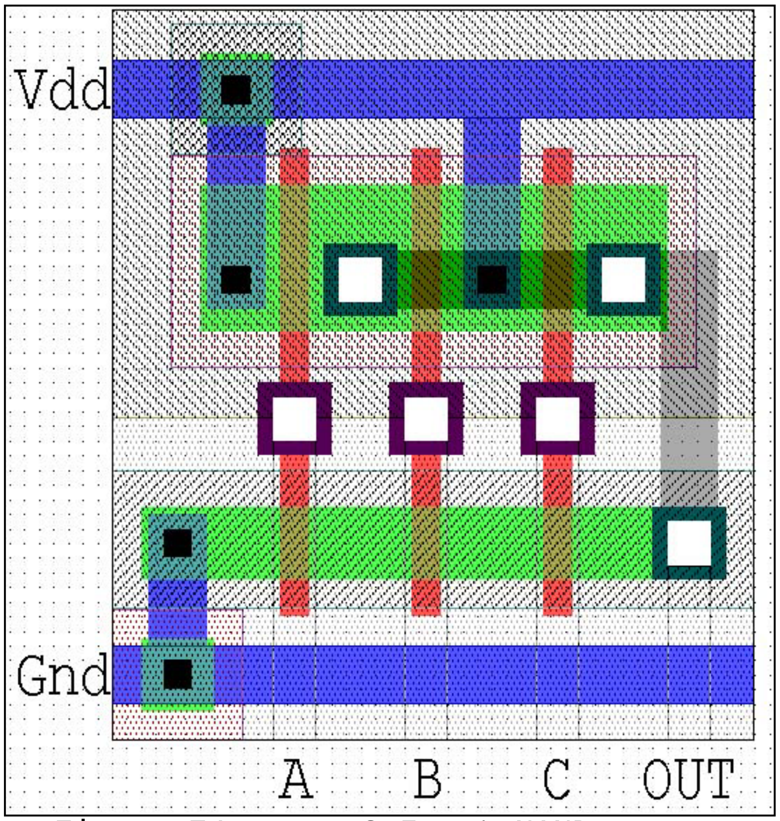


Figure 74. 3-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND3      Version 1.24
* Extract Date and Time:  08/21/2001 - 17:31

.SUBCKT caa_NAND3 A B C Gnd OUT Vdd

M1 OUT C 8 Gnd CMOSN L=180n W=450n
M2 8 B 7 Gnd CMOSN L=180n W=450n
M3 7 A Gnd Gnd CMOSN L=180n W=450n
M4 OUT C Vdd Vdd CMOSP L=180n W=900n
M5 Vdd B OUT Vdd CMOSP L=180n W=900n
M6 OUT A Vdd Vdd CMOSP L=180n W=900n

* Total Nodes: 8
* Total Elements: 6
.ENDS
```

(2)   3-Input NAND For Group GP

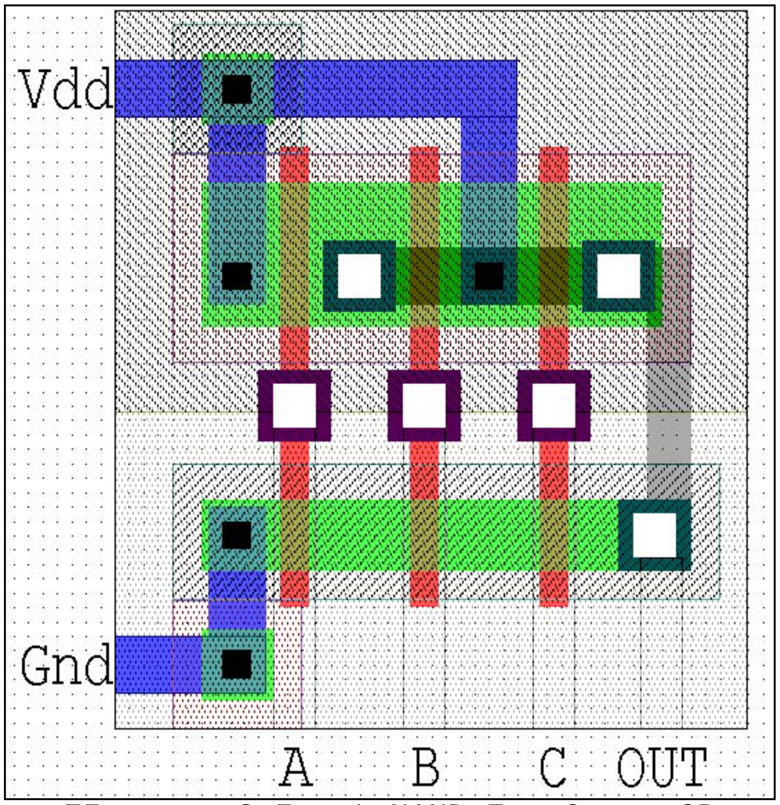


Figure 75.   3-Input NAND For Group GP.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23 * Cell:
caa_NAND3_GrpG    Version 1.25
* Extract Date and Time:  08/21/2001 - 14:17

.SUBCKT caa_NAND3_GrpG A B C Gnd OUT Vdd

M1 OUT C 8 Gnd CMOSN L=180n W=450n
M2 8 B 7 Gnd CMOSN L=180n W=450n
M3 7 A Gnd Gnd CMOSN L=180n W=450n
M4 OUT C Vdd Vdd CMOSP L=180n W=900n
M5 Vdd B OUT Vdd CMOSP L=180n W=900n
M6 OUT A Vdd Vdd CMOSP L=180n W=900n

* Total Nodes: 8
* Total Elements: 6
.ENDS
```

c. 4-Input

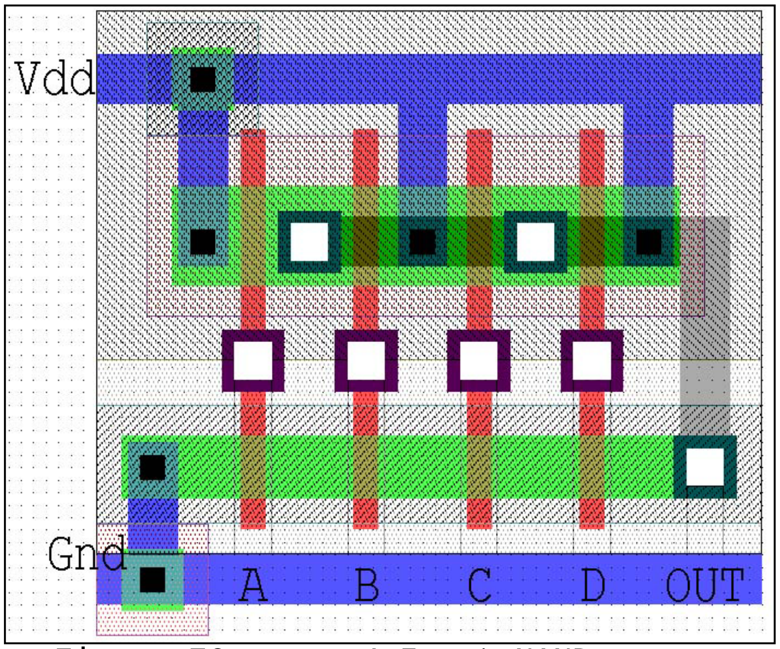


Figure 76. 4-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND4      Version 1.26
* Extract Date and Time:  07/23/2001 - 20:40

.SUBCKT caa_NAND4 A B C D Gnd OUT Vdd

M1 9 C 10 Gnd CMOSN L=180n W=450n
M2 OUT B 9 Gnd CMOSN L=180n W=450n
M3 10 D 8 Gnd CMOSN L=180n W=450n
M4 8 A Gnd Gnd CMOSN L=180n W=450n
M5 OUT C Vdd Vdd CMOSP L=180n W=720n
M6 Vdd D OUT Vdd CMOSP L=180n W=720n
M7 Vdd B OUT Vdd CMOSP L=180n W=720n
M8 OUT A Vdd Vdd CMOSP L=180n W=720n

* Total Nodes: 10
* Total Elements: 8
.ENDS
```

d. 5-Input

(1) 5-Input NAND

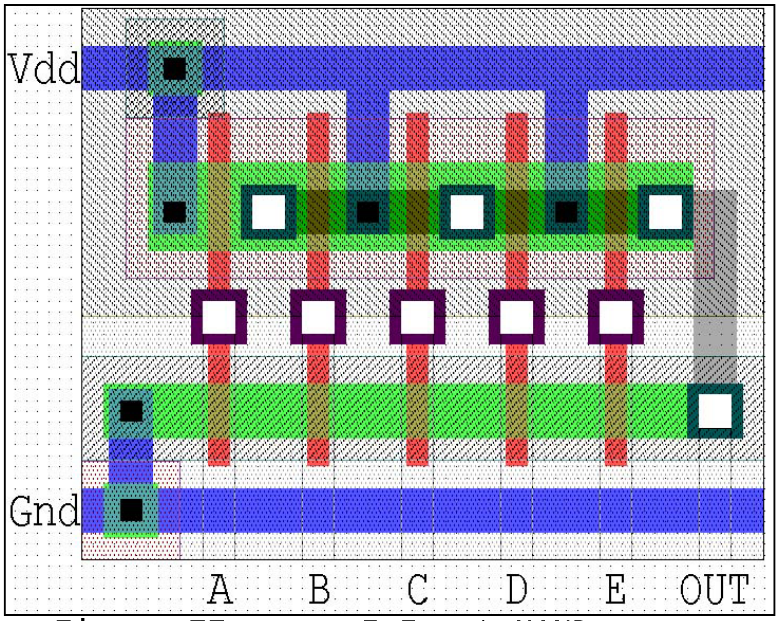


Figure 77. 5-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND5      Version 1.25
* Extract Date and Time:  08/21/2001 - 17:32

.SUBCKT caa_NAND5 A B C D E Gnd OUT Vdd

M1 OUT E 12 Gnd CMOSN L=180n W=450n
M2 12 D 11 Gnd CMOSN L=180n W=450n
M3 11 C 10 Gnd CMOSN L=180n W=450n
M4 10 B 9 Gnd CMOSN L=180n W=450n
M5 9 A Gnd Gnd CMOSN L=180n W=450n
M6 OUT E Vdd Vdd CMOSP L=180n W=720n
M7 Vdd D OUT Vdd CMOSP L=180n W=720n
M8 OUT C Vdd Vdd CMOSP L=180n W=720n
M9 Vdd B OUT Vdd CMOSP L=180n W=720n
M10 OUT A Vdd Vdd CMOSP L=180n W=720n

* Total Nodes: 12
* Total Elements: 10
.ENDS
```

(2) 5-Input NAND For Group GP

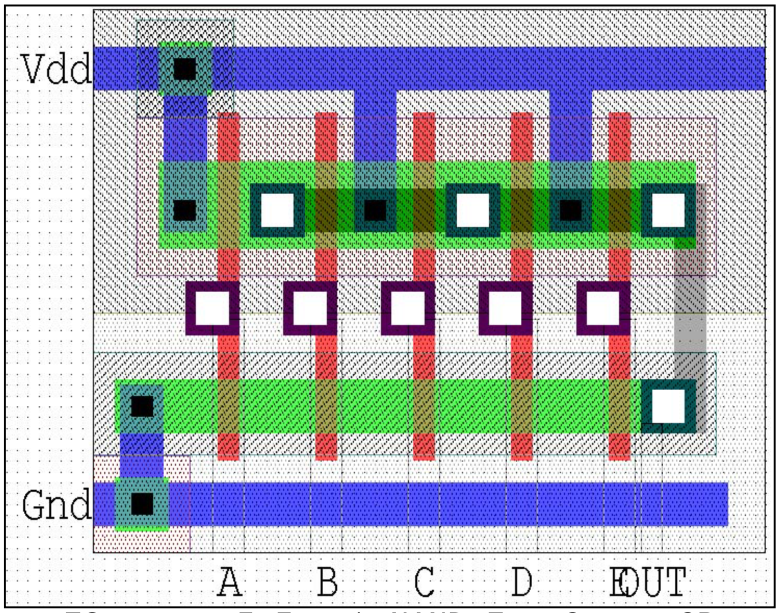


Figure 78. 5-Input NAND For Group GP.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND5_GrpG Version 1.29
* Extract Date and Time:  08/21/2001 - 14:16

.SUBCKT caa_NAND5_GrpG A B C D E Gnd OUT Vdd

M1 12 A Gnd Gnd CMOSN L=180n W=450n
M2 11 B 12 Gnd CMOSN L=180n W=450n
M3 10 C 11 Gnd CMOSN L=180n W=450n
M4 9 D 10 Gnd CMOSN L=180n W=450n
M5 OUT E 9 Gnd CMOSN L=180n W=450n
M6 OUT A Vdd Vdd CMOSP L=180n W=720n
M7 Vdd B OUT Vdd CMOSP L=180n W=720n
M8 OUT C Vdd Vdd CMOSP L=180n W=720n
M9 Vdd D OUT Vdd CMOSP L=180n W=720n
M10 OUT E Vdd Vdd CMOSP L=180n W=720n

* Total Nodes: 12
* Total Elements: 10
.ENDS
```

e. 6-Input

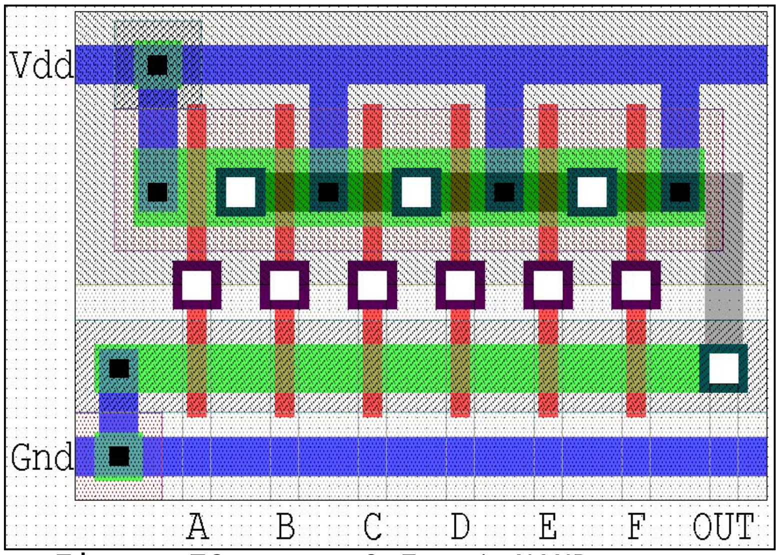


Figure 79. 6-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND6      Version 1.26
* Extract Date and Time:  08/21/2001 - 17:32

.SUBCKT caa_NAND6 A B C D E F Gnd OUT Vdd

M1 13 E 14 Gnd CMOSN L=180n W=450n
M2 OUT F 13 Gnd CMOSN L=180n W=450n
M3 11 C 12 Gnd CMOSN L=180n W=450n
M4 14 D 11 Gnd CMOSN L=180n W=450n
M5 12 B 10 Gnd CMOSN L=180n W=450n
M6 10 A Gnd Gnd CMOSN L=180n W=450n
M7 OUT E Vdd Vdd CMOSP L=180n W=720n
M8 Vdd F OUT Vdd CMOSP L=180n W=720n
M9 OUT C Vdd Vdd CMOSP L=180n W=720n
M10 Vdd D OUT Vdd CMOSP L=180n W=720n
M11 Vdd B OUT Vdd CMOSP L=180n W=720n
M12 OUT A Vdd Vdd CMOSP L=180n W=720n

* Total Nodes: 14
* Total Elements: 12
.ENDS
```

f. 8-Input

(1) 8-Input NAND

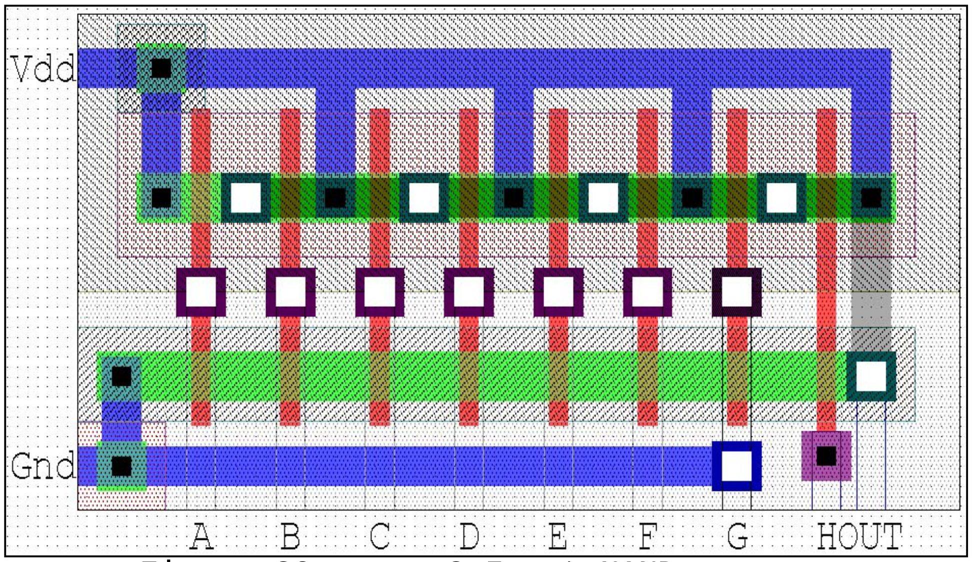


Figure 80. 8-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND8      Version 1.28

.SUBCKT caa_NAND8 A B C D E F G Gnd H OUT Vdd

M1 OUT H 18 Gnd CMOSN L=180n W=450n
M2 18 G 17 Gnd CMOSN L=180n W=450n
M3 17 F 16 Gnd CMOSN L=180n W=450n
M4 16 E 15 Gnd CMOSN L=180n W=450n
M5 13 C 14 Gnd CMOSN L=180n W=450n
M6 15 D 13 Gnd CMOSN L=180n W=450n
M7 14 B 12 Gnd CMOSN L=180n W=450n
M8 12 A Gnd Gnd CMOSN L=180n W=450n
M9 Vdd H OUT Vdd CMOSP L=180n W=450n
M10 OUT G Vdd Vdd CMOSP L=180n W=450n
M11 Vdd F OUT Vdd CMOSP L=180n W=450n
M12 OUT E Vdd Vdd CMOSP L=180n W=450n
M13 OUT C Vdd Vdd CMOSP L=180n W=450n
M14 Vdd D OUT Vdd CMOSP L=180n W=450n
M15 Vdd B OUT Vdd CMOSP L=180n W=450n
M16 OUT A Vdd Vdd CMOSP L=180n W=450n

* Total Nodes: 18
* Total Elements: 16
.ENDS
```

(2) 8-Input NAND For Overflow

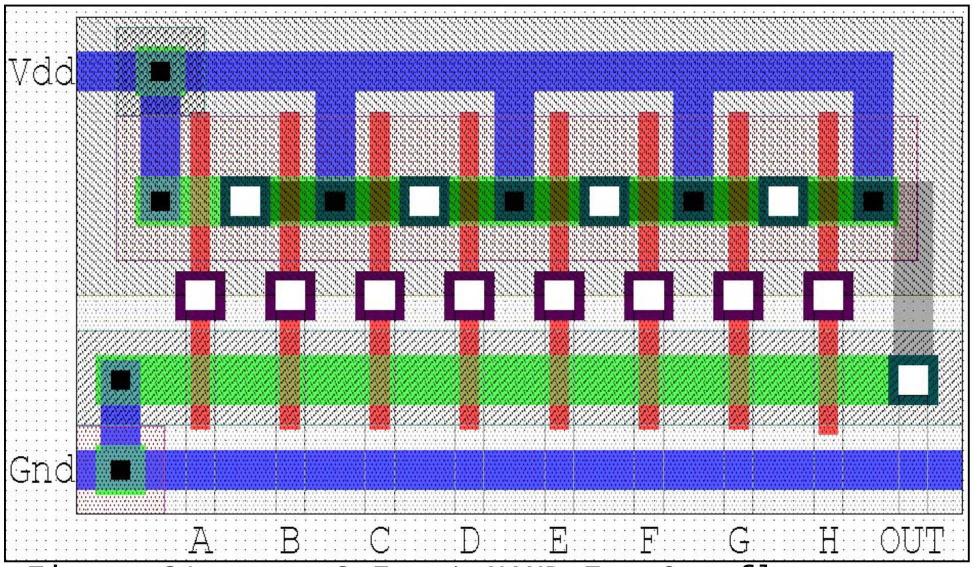


Figure 81. 8-Input NAND For Overflow.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND8_Cis1 Version 1.26
* Extract Date and Time:  08/21/2001 - 14:15

.SUBCKT caa_NAND8_Cis1 A B C D E F G Gnd H OUT Vdd

M1 18 A Gnd Gnd CMOSN L=180n W=450n
M2 17 B 18 Gnd CMOSN L=180n W=450n
M3 15 D 16 Gnd CMOSN L=180n W=450n
M4 16 C 17 Gnd CMOSN L=180n W=450n
M5 14 E 15 Gnd CMOSN L=180n W=450n
M6 13 F 14 Gnd CMOSN L=180n W=450n
M7 12 G 13 Gnd CMOSN L=180n W=450n
M8 OUT H 12 Gnd CMOSN L=180n W=450n
M9 OUT A Vdd Vdd CMOSP L=180n W=450n
M10 Vdd B OUT Vdd CMOSP L=180n W=450n
M11 Vdd D OUT Vdd CMOSP L=180n W=450n
M12 OUT C Vdd Vdd CMOSP L=180n W=450n
M13 OUT E Vdd Vdd CMOSP L=180n W=450n
M14 Vdd F OUT Vdd CMOSP L=180n W=450n
M15 OUT G Vdd Vdd CMOSP L=180n W=450n
M16 Vdd H OUT Vdd CMOSP L=180n W=450n

* Total Nodes: 18
* Total Elements: 16
.ENDS
```

g. 9-Input

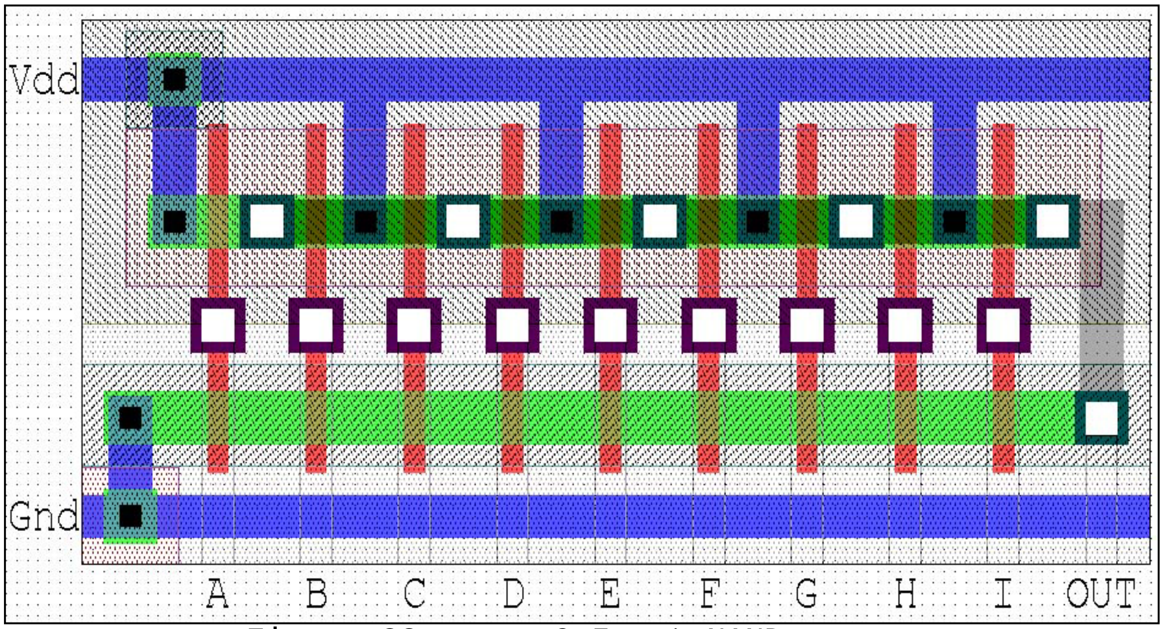


Figure 82. 9-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND9      Version 1.25
* Extract Date and Time:  08/21/2001 - 17:29

.SUBCKT caa_NAND9 A B C D E F G Gnd H I OUT Vdd

M1 OUT I 20 Gnd CMOSN L=180n W=450n
M2 20 H 19 Gnd CMOSN L=180n W=450n
M3 19 G 18 Gnd CMOSN L=180n W=450n
M4 18 F 17 Gnd CMOSN L=180n W=450n
M5 17 E 16 Gnd CMOSN L=180n W=450n
M6 16 D 15 Gnd CMOSN L=180n W=450n
M7 15 C 14 Gnd CMOSN L=180n W=450n
M8 14 B 13 Gnd CMOSN L=180n W=450n
M9 13 A Gnd Gnd CMOSN L=180n W=450n
M10 OUT I Vdd Vdd CMOSP L=180n W=450n
M11 Vdd H OUT Vdd CMOSP L=180n W=450n
M12 OUT G Vdd Vdd CMOSP L=180n W=450n
M13 Vdd F OUT Vdd CMOSP L=180n W=450n
M14 OUT E Vdd Vdd CMOSP L=180n W=450n
M15 Vdd D OUT Vdd CMOSP L=180n W=450n
M16 OUT C Vdd Vdd CMOSP L=180n W=450n
M17 Vdd B OUT Vdd CMOSP L=180n W=450n
M18 OUT A Vdd Vdd CMOSP L=180n W=450n

* Total Nodes: 20
* Total Elements: 18
.ENDS
```

h. 13-Input

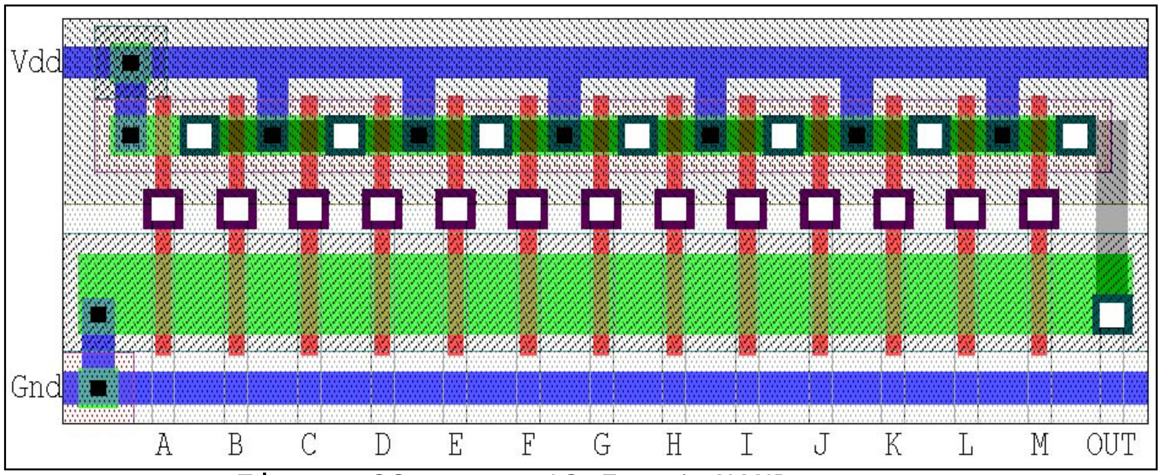


Figure 83. 13-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND13     Version 1.26

.SUBCKT caa_NAND13 A B C D E F G Gnd H I J K L M OUT Vdd

M1 OUT M 5 Gnd CMOSN L=180n W=900n
M2 5 L 4 Gnd CMOSN L=180n W=900n
M3 4 K 27 Gnd CMOSN L=180n W=900n
M4 OUT M Vdd Vdd CMOSP L=180n W=450n
M5 Vdd L OUT Vdd CMOSP L=180n W=450n
M6 OUT K Vdd Vdd CMOSP L=180n W=450n
M7 27 J 28 Gnd CMOSN L=180n W=900n
M8 28 I 26 Gnd CMOSN L=180n W=900n
M9 26 H 25 Gnd CMOSN L=180n W=900n
M10 25 G 24 Gnd CMOSN L=180n W=900n
M11 24 F 23 Gnd CMOSN L=180n W=900n
M12 23 E 22 Gnd CMOSN L=180n W=900n
M13 22 D 21 Gnd CMOSN L=180n W=900n
M14 21 C 20 Gnd CMOSN L=180n W=900n
M15 20 B 19 Gnd CMOSN L=180n W=900n
M16 19 A Gnd Gnd CMOSN L=180n W=900n
M17 Vdd J OUT Vdd CMOSP L=180n W=450n
M18 OUT I Vdd Vdd CMOSP L=180n W=450n
M19 Vdd H OUT Vdd CMOSP L=180n W=450n
M20 OUT G Vdd Vdd CMOSP L=180n W=450n
M21 Vdd F OUT Vdd CMOSP L=180n W=450n
M22 OUT E Vdd Vdd CMOSP L=180n W=450n
M23 Vdd D OUT Vdd CMOSP L=180n W=450n
M24 OUT C Vdd Vdd CMOSP L=180n W=450n
M25 Vdd B OUT Vdd CMOSP L=180n W=450n
M26 OUT A Vdd Vdd CMOSP L=180n W=450n
* Total Nodes: 28
* Total Elements: 26
.ENDS
```

i. 15-Input

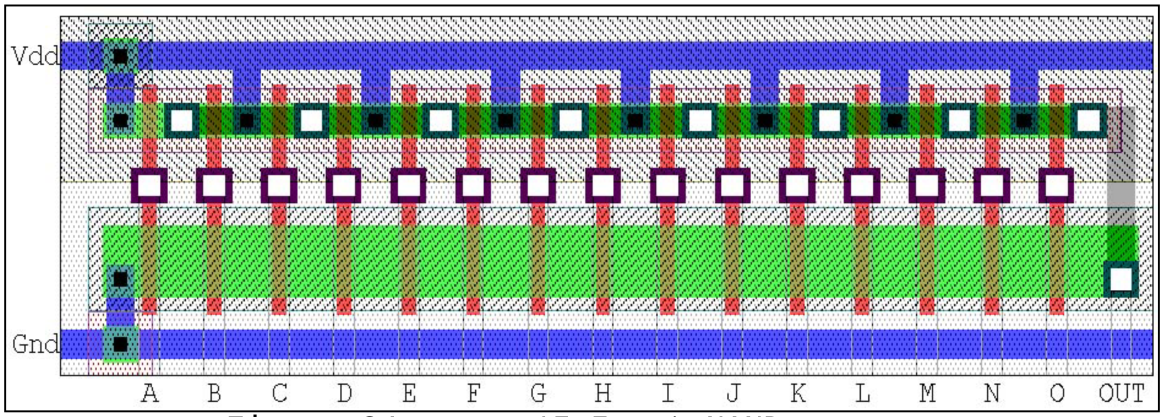


Figure 84. 15-Input NAND.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NAND15     Version 1.28

.SUBCKT caa_NAND15 A B C D E F G Gnd H I J K L M N O OUT Vdd

M1 OUT O 9 Gnd CMOSN L=180n W=900n
M2 9 N 8 Gnd CMOSN L=180n W=900n
M3 8 M 7 Gnd CMOSN L=180n W=900n
M4 7 L 6 Gnd CMOSN L=180n W=900n
M5 6 K 31 Gnd CMOSN L=180n W=900n
M6 OUT O Vdd Vdd CMOSP L=180n W=450n
M7 Vdd N OUT Vdd CMOSP L=180n W=450n
M8 OUT M Vdd Vdd CMOSP L=180n W=450n
M9 Vdd L OUT Vdd CMOSP L=180n W=450n
M10 OUT K Vdd Vdd CMOSP L=180n W=450n
M11 31 J 32 Gnd CMOSN L=180n W=900n
M12 32 I 30 Gnd CMOSN L=180n W=900n
M13 30 H 29 Gnd CMOSN L=180n W=900n
M14 29 G 28 Gnd CMOSN L=180n W=900n
M15 28 F 27 Gnd CMOSN L=180n W=900n
M16 27 E 26 Gnd CMOSN L=180n W=900n
M17 26 D 25 Gnd CMOSN L=180n W=900n
M18 25 C 24 Gnd CMOSN L=180n W=900n
M19 24 B 23 Gnd CMOSN L=180n W=900n
M20 23 A Gnd Gnd CMOSN L=180n W=900n
M21 Vdd J OUT Vdd CMOSP L=180n W=450n
M22 OUT I Vdd Vdd CMOSP L=180n W=450n
M23 Vdd H OUT Vdd CMOSP L=180n W=450n
M24 OUT G Vdd Vdd CMOSP L=180n W=450n
M25 Vdd F OUT Vdd CMOSP L=180n W=450n
M26 OUT E Vdd Vdd CMOSP L=180n W=450n
M27 Vdd D OUT Vdd CMOSP L=180n W=450n
M28 OUT C Vdd Vdd CMOSP L=180n W=450n
M29 Vdd B OUT Vdd CMOSP L=180n W=450n
M30 OUT A Vdd Vdd CMOSP L=180n W=450n
* Total Nodes: 32
* Total Elements: 30
.ENDS
```

3. NOR

a. 2-Input

(1) 2-Input NOR

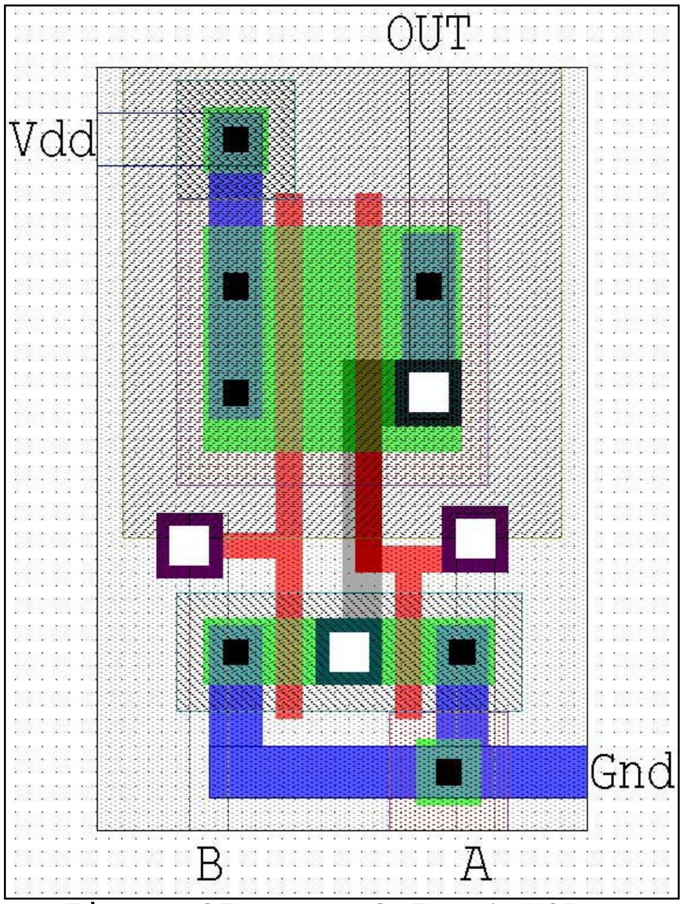


Figure 85. 2-Input NOR.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NOR2 Version 1.30
* Extract Date and Time:  07/25/2001 - 21:11

.SUBCKT caa_NOR2 A B Gnd OUT Vdd

M1 OUT B Gnd Gnd CMOSN L=180n W=450n
M2 Gnd A OUT Gnd CMOSN L=180n W=450n
M3 6 B Vdd Vdd CMOSP L=180n W=1.53u
M4 OUT A 6 Vdd CMOSP L=180n W=1.53u

* Total Nodes: 6
* Total Elements: 4
.ENDS
```

(2) 2-Input NOR For Group GP

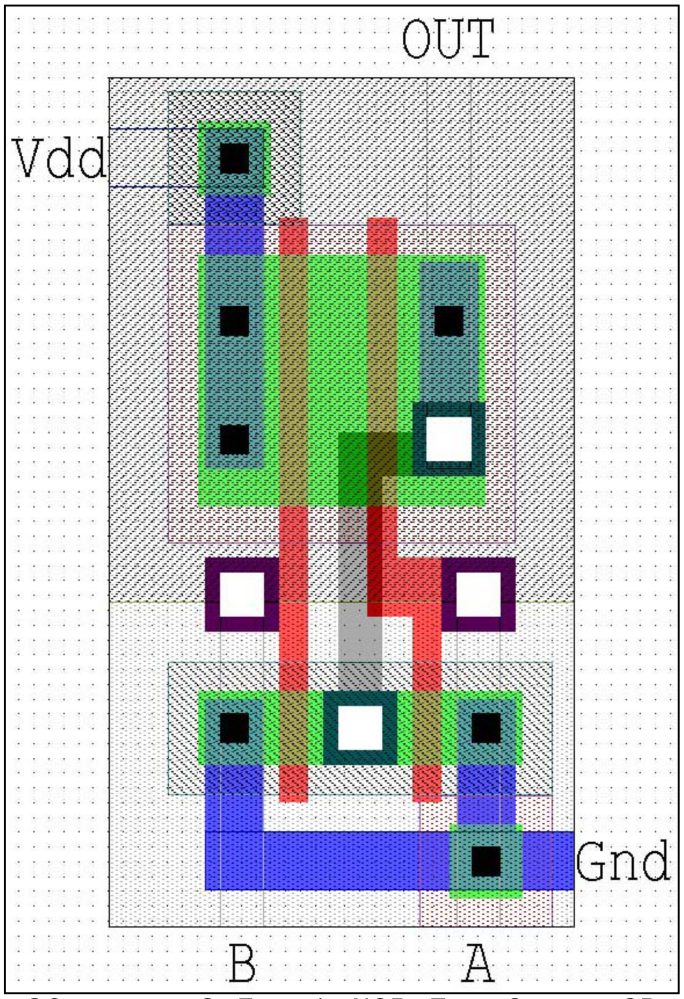


Figure 86. 2-Input NOR For Group GP.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NOR2_GrpP  Version 1.37
* Extract Date and Time:  08/21/2001 - 14:17

.SUBCKT caa_NOR2_GrpP A B Gnd OUT Vdd

M1 Gnd B OUT Gnd CMOSN L=180n W=450n
M2 OUT A Gnd Gnd CMOSN L=180n W=450n
M3 Vdd B 6 Vdd CMOSP L=180n W=1.53u
M4 6 A OUT Vdd CMOSP L=180n W=1.53u

* Total Nodes: 6
* Total Elements: 4
.ENDS
```

b. 4-Input

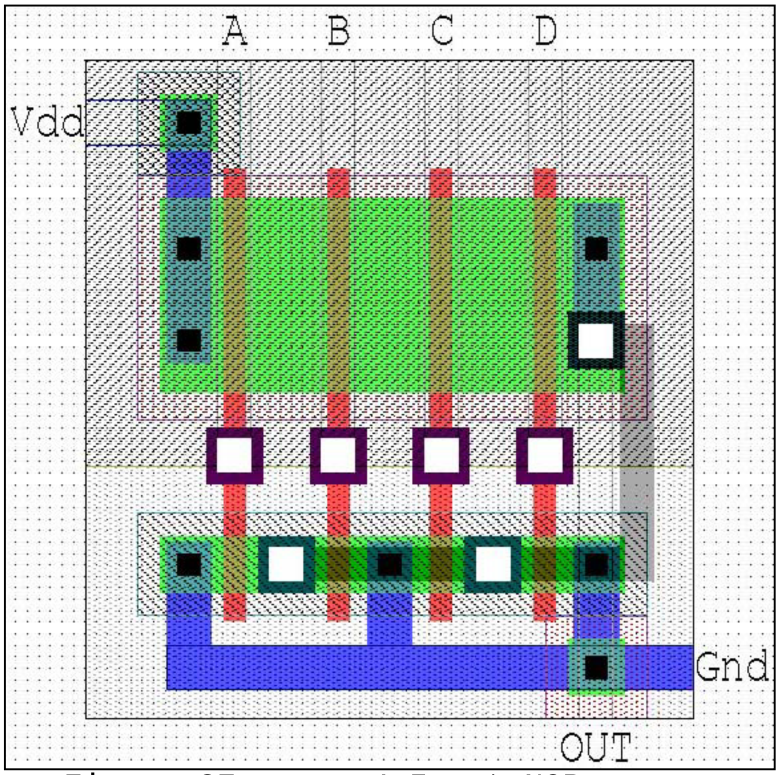


Figure 87. 4-Input NOR.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_NOR4_GrpP  Version 1.37
* Extract Date and Time:  08/18/2001 - 12:44

.SUBCKT caa_NOR4_GrpP A B C D Gnd OUT Vdd

M1 Gnd C OUT Gnd CMOSN L=180n W=450n
M2 OUT D Gnd Gnd CMOSN L=180n W=450n
M3 OUT B Gnd Gnd CMOSN L=180n W=450n
M4 Gnd A OUT Gnd CMOSN L=180n W=450n
M5 9 C 10 Vdd CMOSP L=180n W=1.53u
M6 10 D OUT Vdd CMOSP L=180n W=1.53u
M7 8 B 9 Vdd CMOSP L=180n W=1.53u
M8 Vdd A 8 Vdd CMOSP L=180n W=1.53u

* Total Nodes: 10
* Total Elements: 8
.ENDS
```

B. LEVEL TWO – SECONDARY SUB-CIRCUITS

1. Sum Circuits

a. S0 Circuit

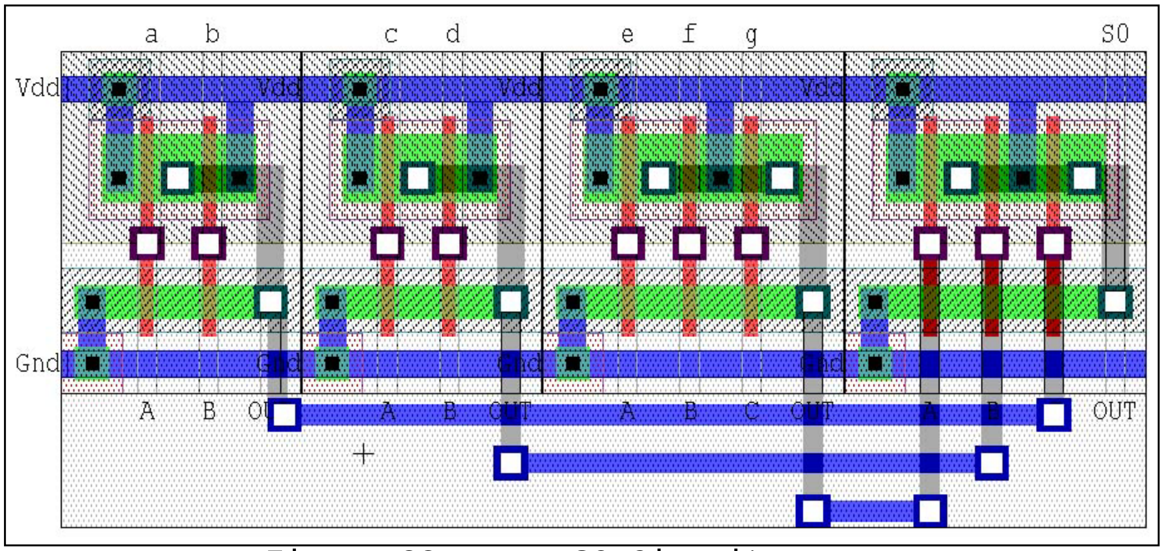


Figure 88. S0 Circuit.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_S0   Version 1.10
* Extract Date and Time:  07/23/2001 - 22:37

.SUBCKT caa_S0 a b c d e f g Gnd S0 Vdd

X1 10 9 8 Gnd S0 Vdd caa_NAND3
X2 e f g Gnd 10 Vdd caa_NAND3
X3 c d Gnd 9 Vdd caa_NAND2
X4 a b Gnd 8 Vdd caa_NAND2

* Total Nodes: 13
* Total Elements: 4
.ENDS
```

b. S1 Circuit

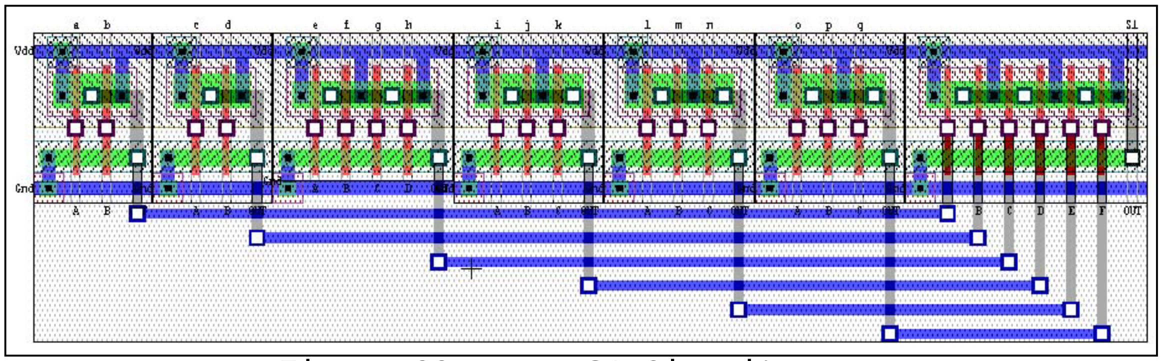


Figure 89. S1 Circuit.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_S1

.SUBCKT caa_S1 a b c d e f g Gnd h i j k l m n o p q S1 Vdd

X1 24 23 22 21 20 19 Gnd S1 Vdd caa_NAND6
X2 o p q Gnd 19 Vdd caa_NAND3
X3 l m n Gnd 20 Vdd caa_NAND3
X4 i j k Gnd 21 Vdd caa_NAND3
X5 e f g h Gnd 22 Vdd caa_NAND4
X6 c d Gnd 23 Vdd caa_NAND2
X7 a b Gnd 24 Vdd caa_NAND2
.ENDS
```

c. S2 Circuit

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_S2

.SUBCKT caa_S2 a b c d e f g Gnd h i j k l m n o p q r s S2 t u v Vdd w
+ x y z za zb zc zd

X1 40 39 38 37 36 35 34 Gnd 33 32 S2 Vdd caa_NAND9
X2 za zb zc zd Gnd 32 Vdd caa_NAND4
X3 w x y z Gnd 33 Vdd caa_NAND4
X4 s t u v Gnd 34 Vdd caa_NAND4
X5 n o p q r Gnd 35 Vdd caa_NAND5
X6 k l m Gnd 36 Vdd caa_NAND3
X7 h i j Gnd 37 Vdd caa_NAND3
X8 f g Gnd 38 Vdd caa_NAND2
X9 a b c Gnd 40 Vdd caa_NAND3
X10 d e Gnd 39 Vdd caa_NAND2
.ENDS
```

d. S3 Circuit

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_S3

.SUBCKT caa_S3 a b c d e f g Gnd h i j k l m n o p q r s S3 t u v Vdd w
+ x y z za zb zc zd ze zf zg zh zi zj zk zl zm zn zo zp zq zr zs zt zu
+ zv zw zx zy zz zza zzb zzc

X1 69 68 67 66 65 64 63 Gnd 62 61 60 59 58 57 S3 Vdd caa_NAND13
X2 zy zz zza zzb zzc Gnd 69 Vdd caa_NAND5
X3 zt zu zv zw zx Gnd 68 Vdd caa_NAND5
X4 zo zp zq zr zs Gnd 67 Vdd caa_NAND5
X5 zd ze zf zg zh zi Gnd 65 Vdd caa_NAND6
X6 zj zk zl zm zn Gnd 66 Vdd caa_NAND5
X7 z za zb zc Gnd 64 Vdd caa_NAND4
X8 w x y Gnd 63 Vdd caa_NAND3
X9 s t u v Gnd 62 Vdd caa_NAND4
X10 o p q r Gnd 61 Vdd caa_NAND4
X11 h i j k Gnd 59 Vdd caa_NAND4
X12 d e f g Gnd 58 Vdd caa_NAND4
X13 l m n Gnd 60 Vdd caa_NAND3
X14 a b c Gnd 57 Vdd caa_NAND3
.ENDS
```

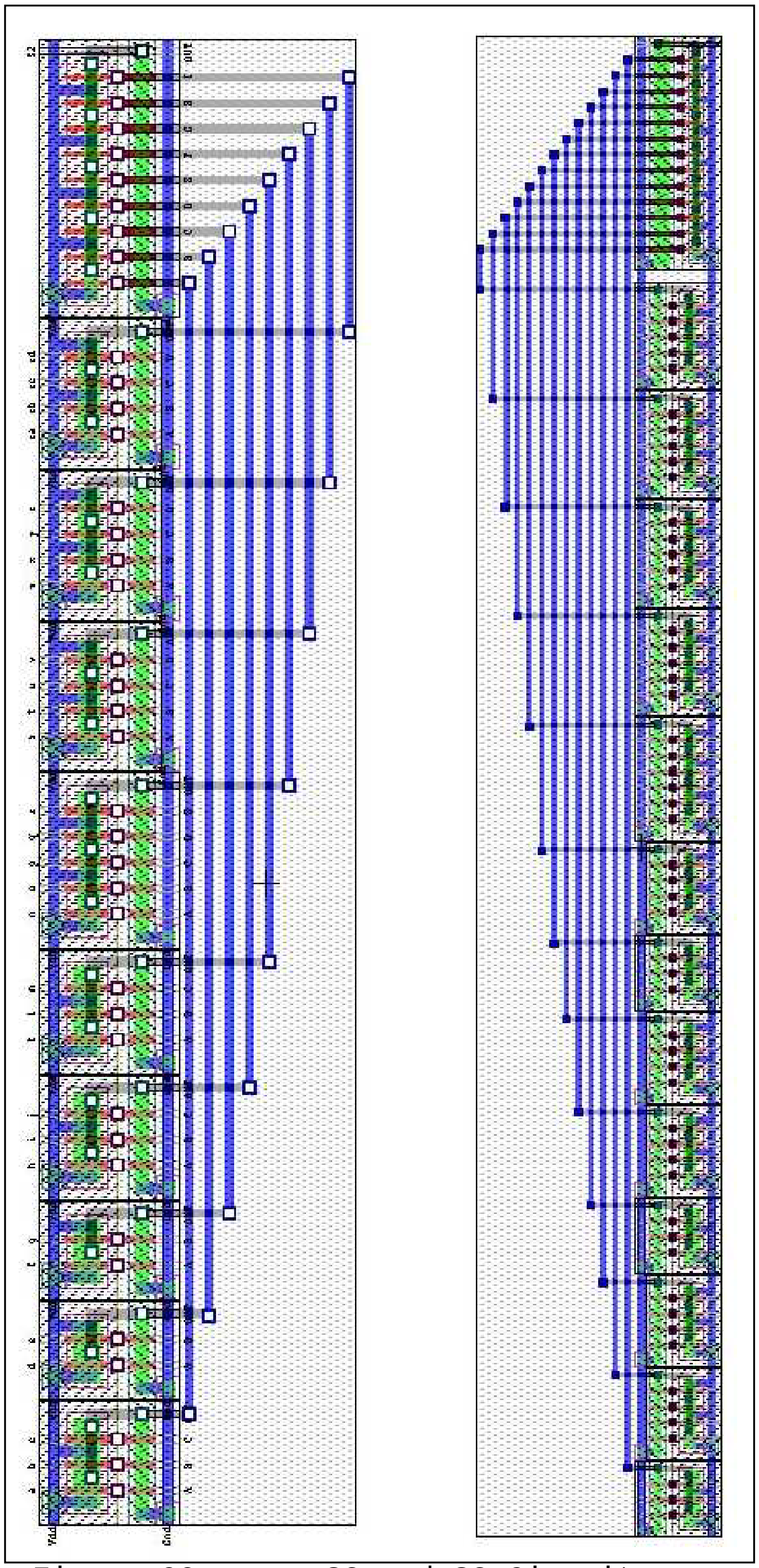


Figure 90.   S2 and S3 Circuits.

2. 4-Bit ALU Group Generate and Propagate (Grp GP)

a. Logic Group GP

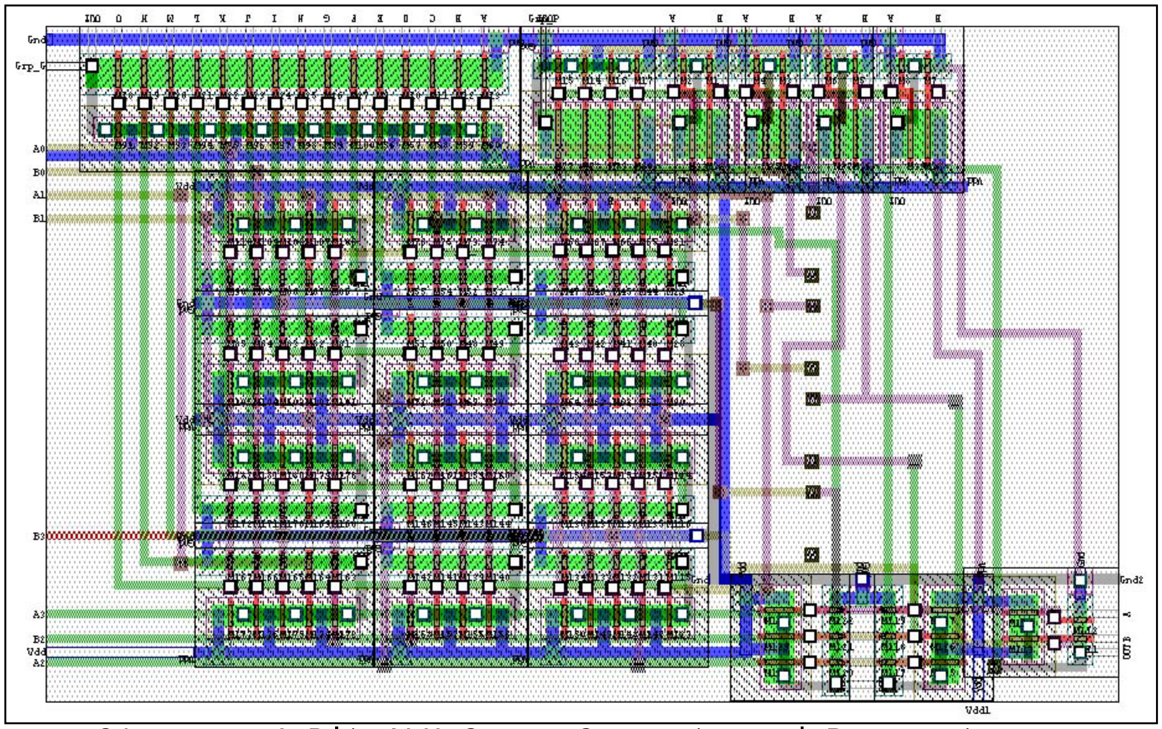


Figure 91. 4-Bit ALU Group Generate and Propagate.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_Grp_GP     Version 1.97

.SUBCKT caa_Grp_GP A0 A1 A2 A3 B0 B1 B2 B3 Gnd Gnd2 Grp_G Grp_P Vdd Vdd1

X1 26 25 24 23 22 21 20 Gnd 18 19 17 16 15 11 13 12 Grp_G Vdd caa_NAND15
X2 A0 B0 B3 A2 A1 Gnd2 12 Vdd caa_NAND5_GrpG
X3 A0 B0 B3 A2 B1 Gnd2 17 Vdd caa_NAND5_GrpG
X4 A0 B0 A3 A2 B1 Gnd2 20 Vdd caa_NAND5_GrpG
X5 A0 B0 A3 A2 A1 Gnd2 23 Vdd caa_NAND5_GrpG
X6 B2 B1 B3 A1 Gnd2 11 Vdd caa_NAND4
X7 A0 B0 B3 A1 B2 Gnd2 13 Vdd caa_NAND5
X8 A2 B1 B3 A1 Gnd2 16 Vdd caa_NAND4
X9 A0 B0 B3 B1 B2 Gnd2 15 Vdd caa_NAND5
X10 A2 B1 A3 A1 Gnd2 19 Vdd caa_NAND4
X11 A0 B0 A3 B1 B2 Gnd2 18 Vdd caa_NAND5
X12 B2 B1 A3 A1 Gnd2 22 Vdd caa_NAND4
X13 A3 B3 Gnd2 26 Vdd1 caa_NAND2_GrpG
X14 B3 B2 A2 Gnd2 25 Vdd1 caa_NAND3_GrpG
X15 A3 B2 A2 Gnd2 24 Vdd caa_NAND3_GrpG
X16 A0 B0 A3 A1 B2 Gnd2 21 Vdd caa_NAND5
X17 27 28 29 9 Gnd Grp_P Vdd caa_NOR4_GrpP
X18 B1 A1 Gnd 27 Vdd caa_NOR2_GrpP
X19 B0 A0 Gnd 28 Vdd caa_NOR2_GrpP
```

```
X20 B2 A2 Gnd 29 Vdd caa_NOR2_GrpP
X21 B3 A3 Gnd 9 Vdd caa_NOR2_GrpP

* Total Nodes: 33
* Total Elements: 21
.ENDS
```

b. Combinational Gate Group GP

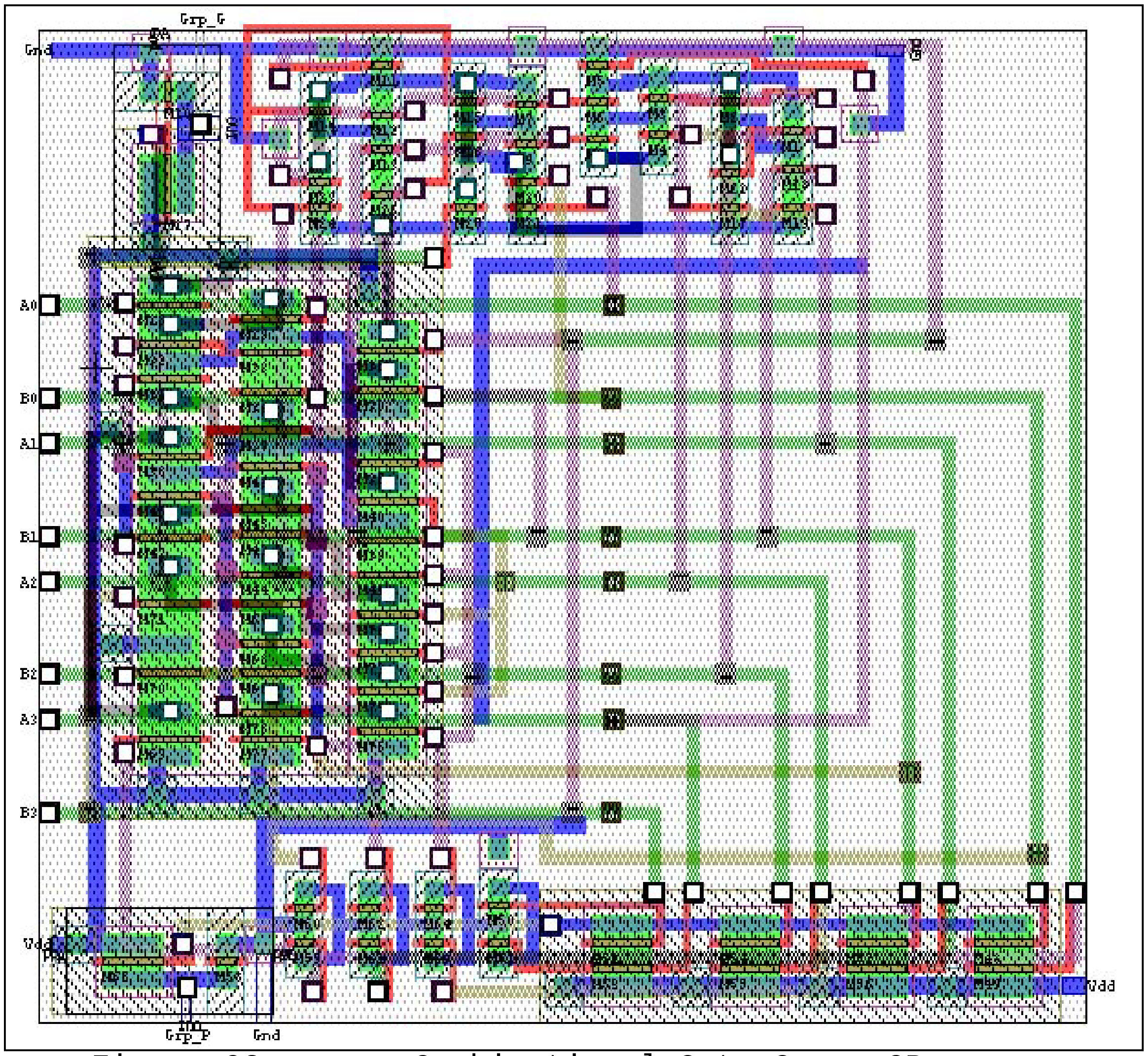


Figure 92. Combinational Gate Group GP.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_Grp_GP_combo     Version 1.70

.SUBCKT caa_Grp_GP_combo A0 A1 A2 A3 B0 B1 B2 B3 Gnd Grp_G Grp_P Vdd

M1 11 B2 5 Gnd CMOSN L=180n W=450n
M2 11 A2 5 Gnd CMOSN L=180n W=450n
M3 1 A2 5 Gnd CMOSN L=180n W=450n
M4 7 A1 1 Gnd CMOSN L=180n W=450n
M5 5 A3 Gnd Gnd CMOSN L=180n W=450n
```

```
M6 1 B2 5 Gnd CMOSN L=180n W=450n
M7 4 B2 16 Gnd CMOSN L=180n W=450n
M8 7 B1 1 Gnd CMOSN L=180n W=450n
M9 7 B1 4 Gnd CMOSN L=180n W=450n
M10 Grp_G U333/A Gnd Gnd CMOSN L=180n W=450n
M11 16 B3 Gnd Gnd CMOSN L=180n W=450n
M12 3 B2 16 Gnd CMOSN L=180n W=450n
M13 26 B1 3 Gnd CMOSN L=180n W=450n
M14 3 A2 16 Gnd CMOSN L=180n W=450n
M15 4 A2 16 Gnd CMOSN L=180n W=450n
M16 7 A1 4 Gnd CMOSN L=180n W=450n
M17 U333/A A2 6 Gnd CMOSN L=180n W=450n
M18 10 B1 11 Gnd CMOSN L=180n W=450n
M19 U333/A A1 10 Gnd CMOSN L=180n W=450n
M20 9 B0 7 Gnd CMOSN L=180n W=450n
M21 U333/A A0 9 Gnd CMOSN L=180n W=450n
M22 6 B2 5 Gnd CMOSN L=180n W=450n
M23 27 B2 16 Gnd CMOSN L=180n W=450n
M24 U333/A A2 27 Gnd CMOSN L=180n W=450n
M25 U333/A A3 16 Gnd CMOSN L=180n W=450n
M26 U333/A A1 26 Gnd CMOSN L=180n W=450n
M27 Grp_G U333/A Vdd Vdd CMOSP L=180n W=1.26u
M28 17 A3 U333/A Vdd CMOSP L=180n W=1.26u
M29 20 B2 17 Vdd CMOSP L=180n W=1.26u
M30 20 B1 18 Vdd CMOSP L=180n W=1.26u
M31 18 A1 20 Vdd CMOSP L=180n W=1.26u
M32 20 A2 17 Vdd CMOSP L=180n W=1.26u
M33 24 A2 20 Vdd CMOSP L=180n W=1.26u
M34 18 B2 24 Vdd CMOSP L=180n W=1.26u
M35 18 B3 U333/A Vdd CMOSP L=180n W=1.26u
M36 18 B2 21 Vdd CMOSP L=180n W=1.26u
M37 21 A3 18 Vdd CMOSP L=180n W=1.26u
M38 21 A3 37 Vdd CMOSP L=180n W=1.26u
M39 23 A2 21 Vdd CMOSP L=180n W=1.26u
M40 21 A2 18 Vdd CMOSP L=180n W=1.26u
M41 37 B2 23 Vdd CMOSP L=180n W=1.26u
M42 37 B1 21 Vdd CMOSP L=180n W=1.26u
M43 38 B1 22 Vdd CMOSP L=180n W=1.26u
M44 47 A1 38 Vdd CMOSP L=180n W=1.26u
M45 22 A1 37 Vdd CMOSP L=180n W=1.26u
M46 37 A1 21 Vdd CMOSP L=180n W=1.26u
M47 46 A3 37 Vdd CMOSP L=180n W=1.26u
M48 28 B0 U334/A Vdd_1 CMOSP L=180n W=1.26u
M49 Vdd_1 A0 28 Vdd_1 CMOSP L=180n W=1.26u
M50 44 B3 U334/A Gnd CMOSN L=180n W=450n
M51 U334/A A3 44 Gnd CMOSN L=180n W=450n
M52 32 B3 U334/A Vdd_1 CMOSP L=180n W=1.26u
M53 Vdd_1 A3 32 Vdd_1 CMOSP L=180n W=1.26u
M54 31 B2 U334/A Vdd_1 CMOSP L=180n W=1.26u
M55 Vdd_1 A2 31 Vdd_1 CMOSP L=180n W=1.26u
M56 Vdd_1 A1 30 Vdd_1 CMOSP L=180n W=1.26u
M57 30 B1 U334/A Vdd_1 CMOSP L=180n W=1.26u
M58 Grp_P U334/A Gnd Gnd CMOSN L=180n W=450n
M59 42 A0 Gnd Gnd CMOSN L=180n W=450n
M60 Gnd B0 42 Gnd CMOSN L=180n W=450n
```

```
M61 43 A1 42 Gnd CMOSN L=180n W=450n
M62 42 B1 43 Gnd CMOSN L=180n W=450n
M63 44 A2 43 Gnd CMOSN L=180n W=450n
M64 43 B2 44 Gnd CMOSN L=180n W=450n
M65 Grp_P U334/A Vdd Vdd CMOSP L=180n W=1.26u
M66 39 B1 48 Vdd CMOSP L=180n W=1.26u
M67 46 B1 47 Vdd CMOSP L=180n W=1.26u
M68 48 A1 46 Vdd CMOSP L=180n W=1.26u
M69 Vdd A0 37 Vdd CMOSP L=180n W=1.26u
M70 37 B0 Vdd Vdd CMOSP L=180n W=1.26u
M71 Vdd B3 46 Vdd CMOSP L=180n W=1.26u
M72 38 A2 37 Vdd CMOSP L=180n W=1.26u
M73 46 B2 38 Vdd CMOSP L=180n W=1.26u
M74 39 A2 46 Vdd CMOSP L=180n W=1.26u
M75 Vdd B2 39 Vdd CMOSP L=180n W=1.26u
M76 45 A1 39 Vdd CMOSP L=180n W=1.26u
M77 Vdd B1 45 Vdd CMOSP L=180n W=1.26u

* Total Nodes: 48
* Total Elements: 77
.ENDS
```

3. ALU Generate and Propagate (GP)

a. 1-Bit ALU Generate and Propagate (1-Bit GP)

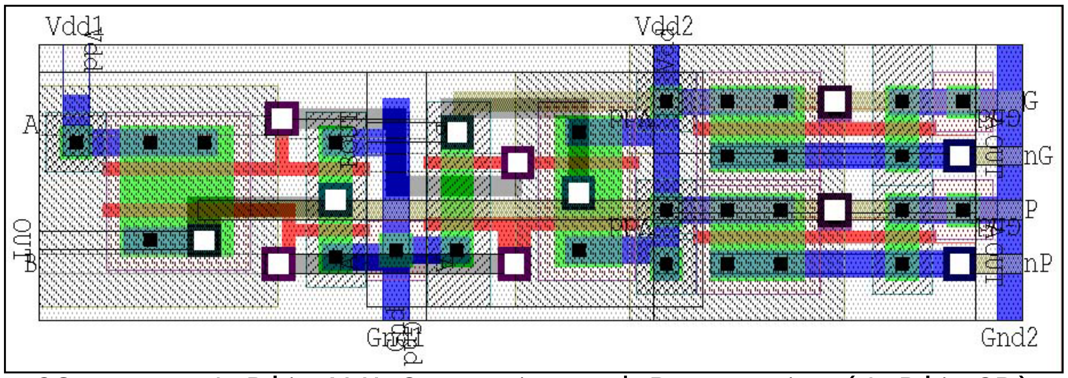


Figure 93. 1-Bit ALU Generate and Propagate (1-Bit GP).

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_PG_Gen     Version 1.16

.SUBCKT caa_PG_Gen A B G Gnd1 Gnd2 nG nP P Vdd1 Vdd2

X1 B A Gnd1 P Vdd1 caa_NOR2
X2 B A Gnd1 G Vdd2 caa_NAND2_H
X3 P Gnd2 nP Vdd2 caa_INV_H
X4 G Gnd2 nG Vdd2 caa_INV_H

* Total Nodes: 10
* Total Elements: 4
.ENDS
```

b. 4-Bit ALU Generate and Propagate (4-Bit GP)

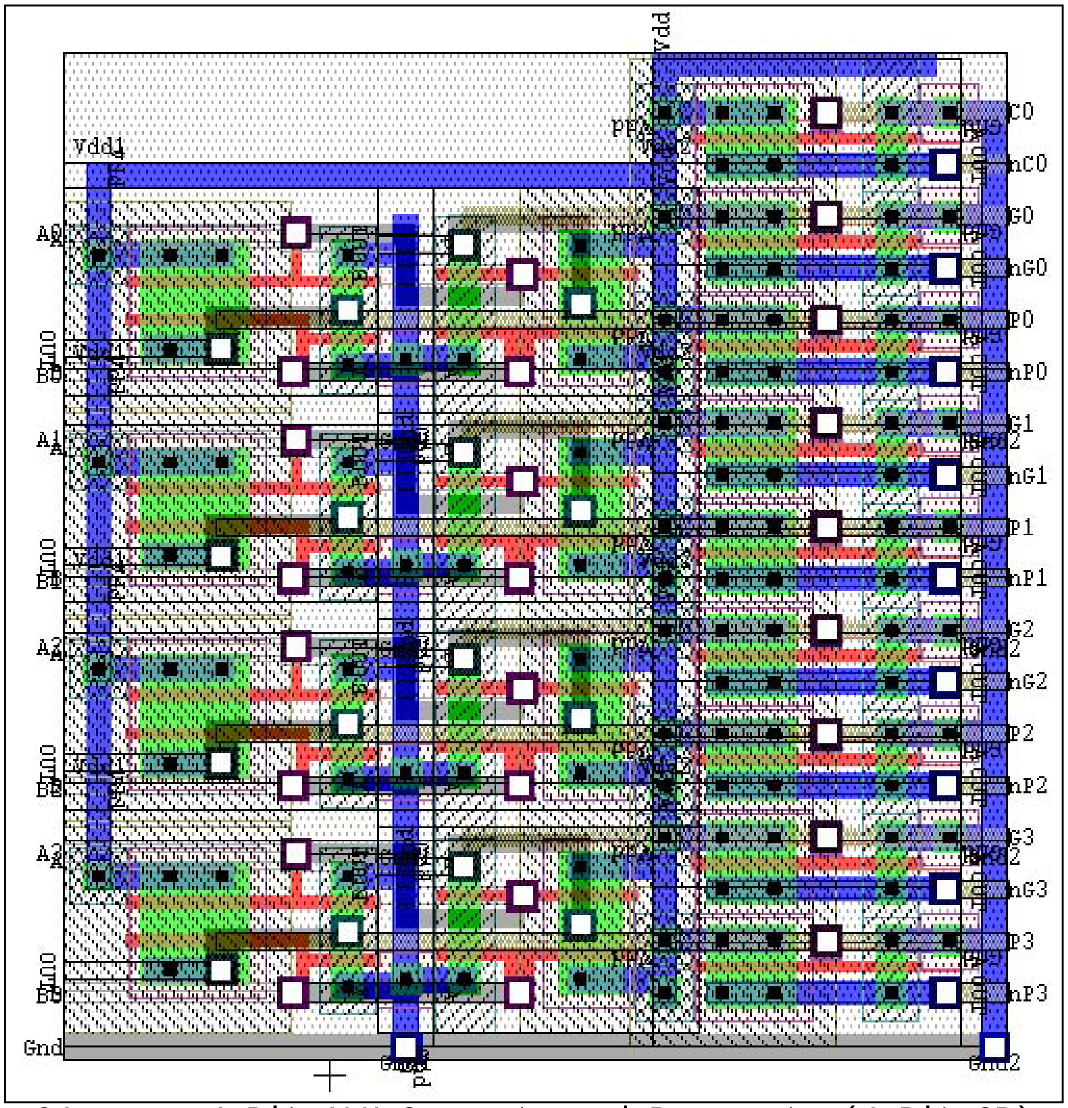


Figure 94. 4-Bit ALU Generate and Propagate (4-Bit GP).

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_PG_Genx4

.SUBCKT caa_PG_Genx4 A0 A1 A2 A3 B0 B1 B2 B3 C0 G0 G1 G2 G3 Gnd nC0 nG0
+ nG1 nG2 nG3 nP0 nP1 nP2 nP3 P0 P1 P2 P3 Vdd

X1 A3 B3 G3 Gnd Gnd nG3 nP3 P3 Vdd Vdd caa_PG_Gen
X2 C0 Gnd nC0 Vdd caa_INV_H
X3 A0 B0 G0 Gnd Gnd nG0 nP0 P0 Vdd Vdd caa_PG_Gen
X4 A1 B1 G1 Gnd Gnd nG1 nP1 P1 Vdd Vdd caa_PG_Gen
X5 A2 B2 G2 Gnd Gnd nG2 nP2 P2 Vdd Vdd caa_PG_Gen
* Total Nodes: 28
* Total Elements: 5
.ENDS
```

4. 4-Bit Overflow

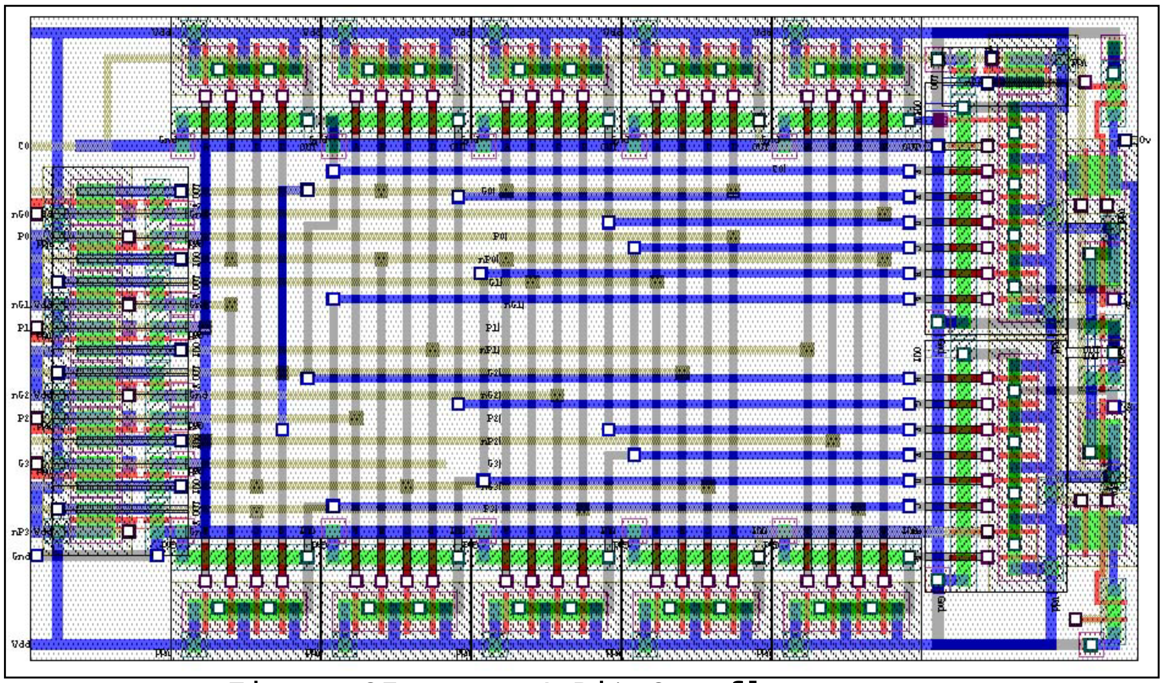


Figure 95. 4-Bit Overflow.

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_overflow

.SUBCKT caa_overflow C0 G3 Gnd nG0 nG1 nG2 nP3 Ov P0 P1 P2 Vdd

X1 34 17 28 27 26 25 24 Gnd 23 32 Vdd caa_NAND8_Cis1
X2 C0 Gnd nC0 Vdd caa_INV_Ov
X3 31 Gnd 20 Vdd caa_INV_Ov
X4 32 Gnd 30 Vdd caa_INV_Ov
X5 28 27 26 25 24 23 18 Gnd 33 31 Vdd caa_NAND8
X6 nP1 nP2 P3 nP0 Gnd 34 Vdd caa_NAND4
X7 G1 G2 nG3 P0 Gnd 17 Vdd caa_NAND4
X8 nP0 G1 nG2 P3 Gnd 26 Vdd caa_NAND4
X9 P2 nP0 nG3 nP1 Gnd 27 Vdd caa_NAND4
X10 nP2 nG1 P3 G0 Gnd 28 Vdd caa_NAND4
X11 nP1 nP2 P3 nG0 Gnd 33 Vdd caa_NAND4
X12 G1 G2 nG3 G0 Gnd 18 Vdd caa_NAND4
X13 G0 G1 nG2 P3 Gnd 25 Vdd caa_NAND4
X14 P2 G0 nG3 nP1 Gnd 24 Vdd caa_NAND4
X15 P1 nP0 nG3 G2 Gnd 23 Vdd caa_NAND4
X16 nP3 Gnd P3 Vdd caa_INV_H
X17 G3 Gnd nG3 Vdd caa_INV_H
X18 P2 Gnd nP2 Vdd caa_INV_H
X19 nG2 Gnd G2 Vdd caa_INV_H
X20 P1 Gnd nP1 Vdd caa_INV_H
X21 nG1 Gnd G1 Vdd caa_INV_H
```

```
X22 P0 Gnd nP0 Vdd caa_INV_H
X23 nG0 Gnd G0 Vdd caa_INV_H
M1 39 30 Ov Gnd CMOSN L=180n W=450n
M2 Gnd C0 39 Gnd CMOSN L=180n W=450n
M3 Ov 20 38 Gnd CMOSN L=180n W=450n
M4 38 nC0 Gnd Gnd CMOSN L=180n W=450n
M5 Ov 30 37 Vdd CMOSP L=180n W=1.26u
M6 37 nC0 Vdd Vdd CMOSP L=180n W=1.26u
M7 Ov 20 36 Vdd CMOSP L=180n W=1.26u
M8 36 C0 Vdd Vdd CMOSP L=180n W=1.26u

* Total Nodes: 39
* Total Elements: 31
.ENDS
```

5. Carry Look-Ahead (CLAH)

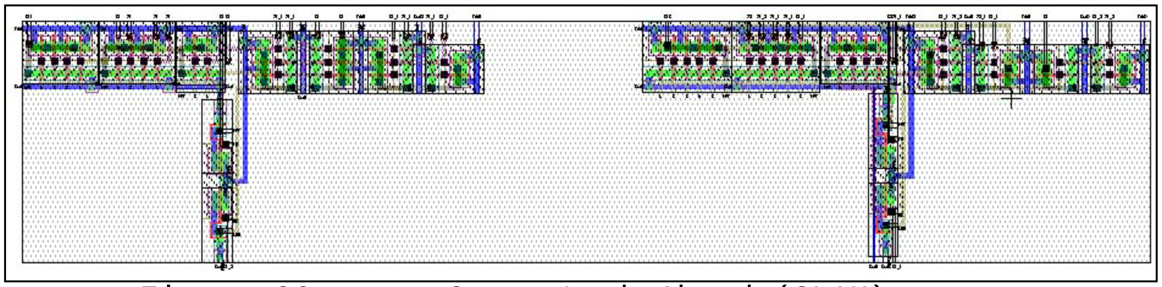


Figure 96. Carry Look-Ahead (CLAH).

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_CLAH

.SUBCKT caa_CLAH C0 C4 C8 C12 C16 C0_1 C0_2 C0_3 G0 G1 G2 G3 G0_1 G0_2
+ G1_1 G1_3 G2_1 Gnd Gnd1 Gnd2 Gnd3 Gnd4 Gnd5 Gnd6 Gnd7 P0 P1 P2 P3 P0_1
+ P0_2 P0_3 P1_1 P1_2 P1_3 P1_4 P2_1 P2_2 P2_3 P3_1 Vdd Vdd1 Vdd2 Vdd3
+ Vdd4 Vdd5

X1 Gnd1 G1_3 46 Vdd caa_INV_CLAH
X2 Gnd G2 43 Vdd caa_INV_CLAH
X3 Gnd6 G0_2 41 Vdd3 caa_INV_CLAH
X4 Gnd7 G3 39 Vdd3 caa_INV_CLAH
X5 39 38 37 36 35 Gnd7 C16 Vdd3 caa_NAND5
X6 P3 P1_3 P2_2 P0_2 C0_2 Gnd7 35 Vdd3 caa_NAND5
X7 45 42 44 43 Gnd C12 Vdd caa_NAND4
X8 P1_4 P2_3 P3_1 G0_2 Gnd7 36 Vdd3 caa_NAND4
X9 G1_1 P2_3 P3_1 Gnd4 38 Vdd3 caa_NAND3
X10 G2_1 P3_1 Gnd4 37 Vdd4 caa_NAND2
X11 P0 P1 P2 C0 Gnd 45 Vdd caa_NAND4
X12 P2_1 P1_1 G0 Gnd2 44 Vdd caa_NAND3
X13 G1 P2_1 Gnd 42 Vdd caa_NAND2
X14 48 47 46 Gnd2 C8 Vdd1 caa_NAND3
X15 C0_1 P0_1 P1_2 Gnd3 48 Vdd1 caa_NAND3
X16 G0_1 P1_2 Gnd3 47 Vdd2 caa_NAND2
X17 40 41 Gnd5 C4 Vdd4 caa_NAND2
X18 P0_3 C0_3 Gnd5 40 Vdd5 caa_NAND2
.ENDS
```

C.   LEVEL THREE – 4-BIT ALU

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_Adderx4

.SUBCKT caa_Adderx4 A0 A1 A2 A3 B0 B1 B2 B3 C0 Gnd Grp_G Grp_P Ov S0 S1
+ S2 S3 Vdd

X1 A0 A1 A2 A3 B0 B1 B2 B3 Gnd Gnd Grp_G Grp_P Vdd Vdd caa_Grp_GP
X2 C0 G3 Gnd nG0 nG1 nG2 nP3 Ov P0 P1 P2 Vdd caa_overflow
X3 A0 A1 A2 A3 B0 B1 B2 B3 C0 G0 G1 G2 G3 Gnd nC0 nG0 nG1
+ nG2 nG3 nP0 nP1 nP2 nP3 P0 P1 P2 P3 Vdd caa_PG_Genx4
X4 nG0 C0 C0 P0 nP0 nC0 G0 Gnd S0 Vdd caa_S0
X5 nG0 P1 nG0 nG1 nP1 G0 G1 Gnd nC0 nP1 P0 G1 C0 nG1 nP0 nP0
+ C0 P1 S1 Vdd caa_S1
X6 nG2 nG0 nP1 P2 nG1 nG2 nG1 Gnd nP1 nG0 P2 nP2 G2 P1 nP2 G2 G1 nC0 G0
+ nG2 S2 nP1 C0 nP0 Vdd nP2 G2 P0 G1 nP1 P2 C0 nP0 caa_S2
X7 nP0 nG2 P3 nP3 G3 G2 P1 Gnd nP2 nP1 nG0 nG3 nG3 nG2 G1 G1nG2 P3 G0 G3
+ S3 nP3 P2 nP1 Vdd P3 nG1 nP2 nP2 G0 nG1 nG3 nC0 nP3 G2 G3 G0 G1 P3 nG0
+ nP1 G3 nP2 C0 nP1 nP0 nP2 nG3 G1 G3 P0 G2 nP3 P3 C0 nP0 nP2 nP1 caa_S3
* Total Nodes: 35
* Total Elements: 7
.ENDS
```

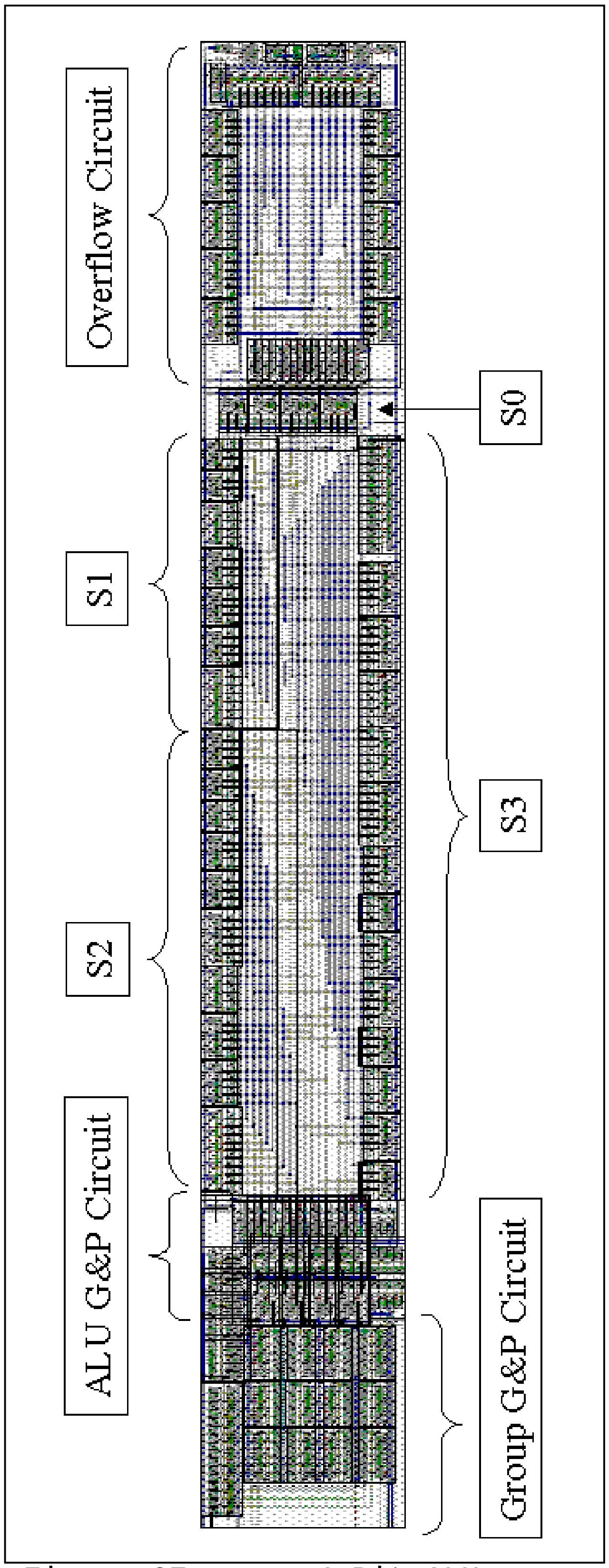


Figure 97. 4-Bit ALU.

D.   LEVEL FOUR – 16-BIT ADDER

```
* Circuit Extracted by Tanner Research's L-Edit Version 8.23
* Cell:  caa_Adderx16
* Extract Date and Time:  08/21/2001 - 13:53

.SUBCKT caa_Adderx16 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11
+ A12 A13 A14 A15 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12
+ B13 B14 B15 C0 C4 C8 C12 C16 Gnd Ov S0 S1 S2 S3 S4 S5 S6
+ S7 S8 S9 S10 S11 S12 S13 S14 S15 Vdd

X1 C0 C4 C8 C12 C16 C0 C0 C0 G0 G1 G2 G3 G0 G0 G1 G1 G2 Gnd
+ Gnd Gnd Gnd Gnd Gnd Gnd Gnd P0 P1 P2 P3 P0 P0 P0 P1 P1 P1
+ P1 P2 P2 P2 P3 Vdd Vdd Vdd Vdd Vdd Vdd caa_CLAH
X2 A8 A9 A10 A11 B8 B9 B10 B11 C8 Gnd G2 P2 1 S8 S9 S10 S11
+ Vdd caa_Adderx4
X3 A4 A5 A6 A7 B4 B5 B6 B7 C4 Gnd G1 P1 2 S4 S5 S6 S7 Vdd
+ caa_Adderx4
X4 A12 A13 A14 A15 B12 B13 B14 B15 C12 Gnd G3 P3 Ov S12 S13
+ S14 S15 Vdd caa_Adderx4
X5 A0 A1 A2 A3 B0 B1 B2 B3 C0 Gnd G0 P0 3 S0 S1 S2 S3 Vdd
+ caa_Adderx4

* Total Nodes: 67
* Total Elements: 5
.ENDS
```

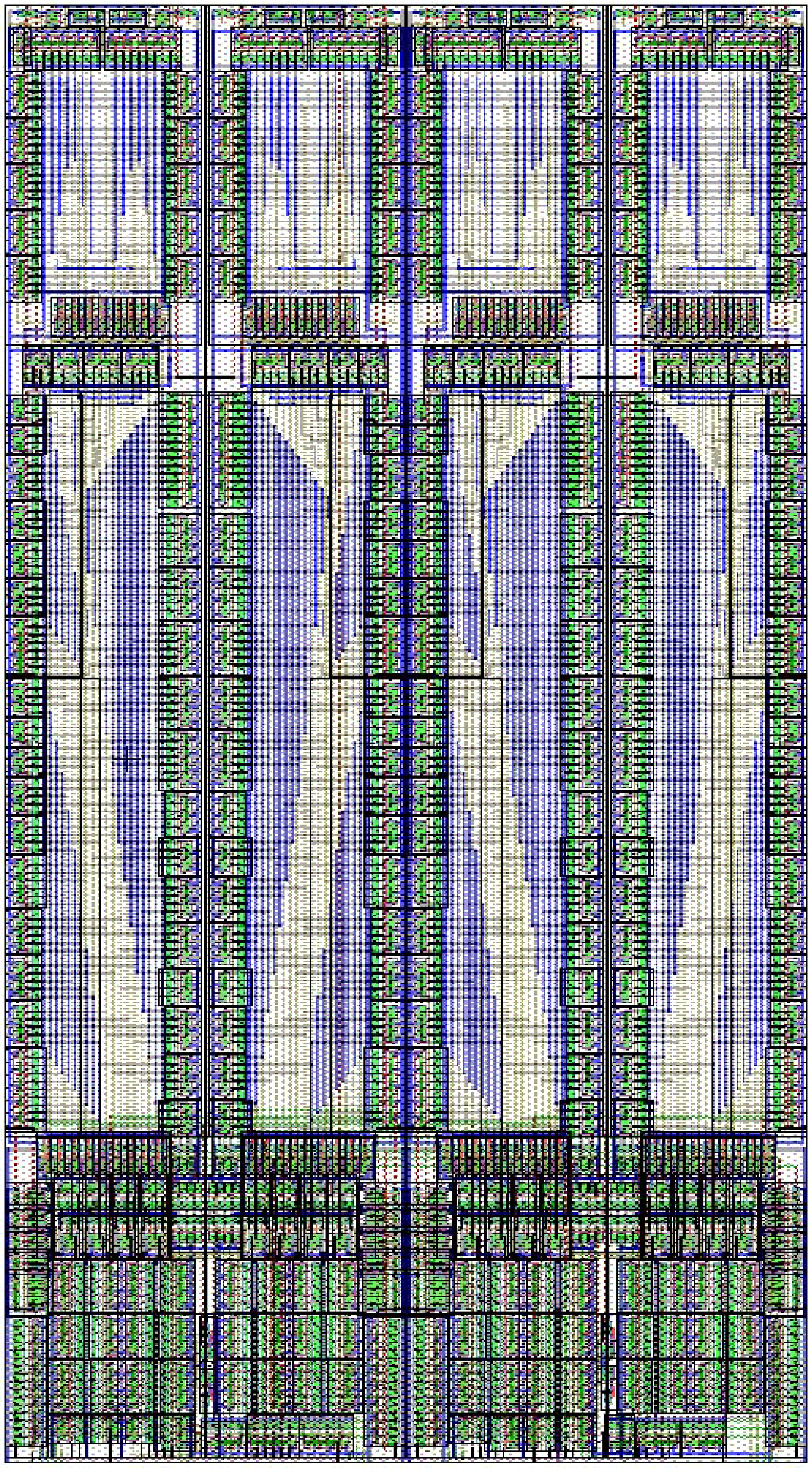


Figure 98. 16-Bit Adder.

APPENDIX C. SIMULATION OF 16-BIT ADDER

A. OVERVIEW

Appendix C contains all the necessary SPICE and MATLAB files to conduct and evaluate one of the simulations used to verify logic functionality and speed.

B. HEADER FILE

```
.INCLUDE TSMC018epi.md
.PARAM lambda=0.09E-06
Vpower Vdd Gnd DC 1.8

.SUBCKT BufInv In Out GND Vdd
M1 Out In Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
+ AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
+ PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 Out In Vdd Vdd CMOSP W=14*lambda L=2*lambda AS=14*lambda*5.5*lambda
+ AD=14*lambda*5.5*lambda PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
+ PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT InputBuf In Out GND Vdd
XInv1 In N1 Gnd Vdd BufInv
XInv2 N1 N2 Gnd Vdd BufInv
XInv3 N2 N3 Gnd Vdd BufInv
XInv4 N3 Out Gnd Vdd BufInv
.ENDS

XBufA0 A0Buf A0 Gnd Vdd InputBuf
XBufA1 A1Buf A1 Gnd Vdd InputBuf
XBufA2 A2Buf A2 Gnd Vdd InputBuf
XBufA3 A3Buf A3 Gnd Vdd InputBuf
XBufA4 A4Buf A4 Gnd Vdd InputBuf
XBufA5 A5Buf A5 Gnd Vdd InputBuf
XBufA6 A6Buf A6 Gnd Vdd InputBuf
XBufA7 A7Buf A7 Gnd Vdd InputBuf
XBufA8 A8Buf A8 Gnd Vdd InputBuf
XBufA9 A9Buf A9 Gnd Vdd InputBuf
XBufA10 A10Buf A10 Gnd Vdd InputBuf
XBufA11 A11Buf A11 Gnd Vdd InputBuf
XBufA12 A12Buf A12 Gnd Vdd InputBuf
XBufA13 A13Buf A13 Gnd Vdd InputBuf
XBufA14 A14Buf A14 Gnd Vdd InputBuf
XBufA15 A15Buf A15 Gnd Vdd InputBuf
XBufB0 B0Buf B0 Gnd Vdd InputBuf
XBufB1 B1Buf B1 Gnd Vdd InputBuf
XBufB2 B2Buf B2 Gnd Vdd InputBuf
XBufB3 B3Buf B3 Gnd Vdd InputBuf
XBufB4 B4Buf B4 Gnd Vdd InputBuf
XBufB5 B5Buf B5 Gnd Vdd InputBuf
```

```
XBufB6 B6Buf B6 Gnd Vdd InputBuf
XBufB7 B7Buf B7 Gnd Vdd InputBuf
XBufB8 B8Buf B8 Gnd Vdd InputBuf
XBufB9 B9Buf B9 Gnd Vdd InputBuf
XBufB10 B10Buf B10 Gnd Vdd InputBuf
XBufB11 B11Buf B11 Gnd Vdd InputBuf
XBufB12 B12Buf B12 Gnd Vdd InputBuf
XBufB13 B13Buf B13 Gnd Vdd InputBuf
XBufB14 B14Buf B14 Gnd Vdd InputBuf
XBufB15 B15Buf B15 Gnd Vdd InputBuf

XBufC0 C0Buf C0 Gnd Vdd InputBuf

VinA0 A0Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA1 A1Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=3.2ns)
VinA2 A2Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=6.4ns)
VinA3 A3Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=12.8ns)
VinA4 A4Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA5 A5Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA6 A6Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA7 A7Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA8 A8Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA9 A9Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA10 A10Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA11 A11Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA12 A12Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA13 A13Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA14 A14Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinA15 A15Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB0 B0Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=25.6ns)
VinB1 B1Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=51.2ns)
VinB2 B2Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=102.4ns)
VinB3 B3Buf Gnd bit ({10} on=1.8 off=0.0 rt=100ps ft=100ps pw=204.8ns)
VinB4 B4Buf Gnd bit ({00} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB5 B5Buf Gnd bit ({00} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB6 B6Buf Gnd bit ({00} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB7 B7Buf Gnd bit ({00} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB8 B8Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB9 B9Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB10 B10Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB11 B11Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB12 B12Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB13 B13Buf Gnd bit ({11} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB14 B14Buf Gnd bit ({10} on=1.8 off=0.0 rt=100ps ft=100ps pw=1.6ns)
VinB15 B15Buf Gnd bit ({1100} on=1.8 off=0.0 rt=100ps ft=100ps
pw=1.6ns)

VinC0 C0Buf Gnd bit ({01} on=1.8 off=0.0 rt=100ps ft=100ps pw=409.6ns)
.TRAN 20PS 819.2NS
.OPTIONS PRTDEL=1.6n
.PRINT TRAN "caa_Adder_625.dat" V(S15) V(S14) V(S13) V(S12) V(S11)
+ V(S10) V(S9) V(S8) V(S7) V(S6) V(S5) V(S4) V(S3) V(S2) V(S1) V(S0)
+ V(C16) V(Ov) V(CLK) V(A15) V(A14) V(A13) V(A12) V(A11) V(A10) V(A9)
+ V(A8) V(A7) V(A6) V(A5) V(A4) V(A3) V(A2) V(A1) V(A0)
+ V(B15) V(B14) V(B13) V(B12) V(B11) V(B10) V(B9) V(B8) V(B7) V(B6)
+ V(B5) V(B4) V(B3) V(B2) V(B1) V(B0) V(C0)
```

C. SPICE NET-LIST

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Aug 23, 2001 at 19:37:19

.INCLUDE caa_Adderx16_625.h
.param lambda=0.09E-6
.SUBCKT caa_INV In Out Gnd Vdd
M1 Out In Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 Out In Vdd Vdd CMOSP W=14*lambda L=2*lambda AS=14*lambda*5.5*lambda
AD=14*lambda*5.5*lambda PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_NAND2 A B NAND2 Gnd Vdd
M1 NAND2 A N1 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 NAND2 B Vdd Vdd CMOSP W=10*lambda L=2*lambda AS=10*lambda*5.5*lambda
AD=10*lambda*5.5*lambda PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M4 NAND2 A Vdd Vdd CMOSP W=10*lambda L=2*lambda AS=10*lambda*5.5*lambda
AD=10*lambda*5.5*lambda PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_NAND3 A B C NAND3 Gnd Vdd
M1 NAND3 A N1 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B N5 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 NAND3 C Vdd Vdd CMOSP W=10*lambda L=2*lambda AS=10*lambda*5.5*lambda
AD=10*lambda*5.5*lambda PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M5 NAND3 B Vdd Vdd CMOSP W=10*lambda L=2*lambda AS=10*lambda*5.5*lambda
AD=10*lambda*5.5*lambda PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M6 NAND3 A Vdd Vdd CMOSP W=10*lambda L=2*lambda AS=10*lambda*5.5*lambda
AD=10*lambda*5.5*lambda PS=10*lambda+10*lambda+5.5*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_NAND4 A B C D NAND4 Gnd Vdd
M1 NAND4 A N1 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
```

```
M2 N1 B N5 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C N8 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N8 D Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 NAND4 A Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M6 NAND4 B Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M7 NAND4 C Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M8 NAND4 D Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_NAND5 A B C D E NAND5 Gnd Vdd
M1 NAND5 A N1 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B N5 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C N8 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N8 D N11 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N11 E Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 NAND5 A Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M7 NAND5 B Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M8 NAND5 C Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M9 NAND5 D Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M10 NAND5 E Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
.ENDS
```

```
.SUBCKT caa_CLAH C0 C4 C8 C12 C16 G0 G1 G2 G3 P0 P1 P2 P3 Gnd Vdd
Xcaa_INV_1 G0 N28 Gnd Vdd caa_INV
Xcaa_INV_2 G1 N27 Gnd Vdd caa_INV
Xcaa_INV_3 G2 N26 Gnd Vdd caa_INV
Xcaa_INV_4 G3 N25 Gnd Vdd caa_INV
Xcaa_NAND2_1 C0 P0 N24 Gnd Vdd caa_NAND2
Xcaa_NAND2_2 G0 P1 N23 Gnd Vdd caa_NAND2
Xcaa_NAND2_3 G1 P2 N22 Gnd Vdd caa_NAND2
Xcaa_NAND2_4 G2 P3 N21 Gnd Vdd caa_NAND2
Xcaa_NAND2_5 N28 N24 C4 Gnd Vdd caa_NAND2
Xcaa_NAND3_1 C0 P1 P0 N18 Gnd Vdd caa_NAND3
Xcaa_NAND3_2 G0 P2 P1 N17 Gnd Vdd caa_NAND3
Xcaa_NAND3_3 G1 P3 P2 N16 Gnd Vdd caa_NAND3
Xcaa_NAND3_4 N27 N23 N18 C8 Gnd Vdd caa_NAND3
Xcaa_NAND4_1 C0 P2 P1 P0 N12 Gnd Vdd caa_NAND4
Xcaa_NAND4_2 G0 P3 P2 P1 N11 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 N26 N22 N17 N12 C12 Gnd Vdd caa_NAND4
Xcaa_NAND5_1 C0 P3 P2 P1 P0 N6 Gnd Vdd caa_NAND5
Xcaa_NAND5_2 N25 N21 N16 N11 N6 C16 Gnd Vdd caa_NAND5
.ENDS

.SUBCKT caa_NAND13 A B C D E F G H I J K L M NAND13 Gnd Vdd
M1 N13 J N15 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M2 N15 K N18 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M3 N18 L N1 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M4 NAND13 A N4 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M5 N4 B N6 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M6 N6 C N8 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M7 N8 D N14 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M8 N14 E N9 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M9 N9 F N17 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M10 N17 G N23 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M11 N23 H N12 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
```

```
M12 N12 I N13 Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M13 N1 M Gnd Gnd CMOSN W=10*lambda L=2*lambda AS=5.5*lambda*10*lambda
AD=5.5*lambda*10*lambda PS=10*lambda+5.5*lambda+10*lambda+5.5*lambda
PD=10*lambda+10*lambda+5.5*lambda+5.5*lambda
M14 NAND13 I Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M15 NAND13 H Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M16 NAND13 G Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M17 NAND13 F Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M18 NAND13 E Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M19 NAND13 D Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M20 NAND13 C Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M21 NAND13 B Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M22 NAND13 A Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M23 NAND13 J Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M24 NAND13 K Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M25 NAND13 L Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M26 NAND13 M Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_NAND6 A B C D E F NAND6 Gnd Vdd
M1 NAND6 A N6 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N6 B N5 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
```

```
M3 N5 C N4 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N4 D N2 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N2 E N1 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 N1 F Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M7 NAND6 E Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M8 NAND6 D Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M9 NAND6 C Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M10 NAND6 B Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M11 NAND6 A Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
M12 NAND6 F Vdd Vdd CMOSP W=8*lambda L=2*lambda AS=8*lambda*5.5*lambda
AD=8*lambda*5.5*lambda PS=8*lambda+8*lambda+5.5*lambda+5.5*lambda
PD=8*lambda+8*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_NAND9 A B C D E F G H I NAND9 Gnd Vdd
M1 NAND9 A N1 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B N5 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C N8 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N8 D N11 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N11 E N14 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 N14 F N10 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M7 N10 G N12 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
```

```
M8 N12 H N15 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M9 N15 I Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M10 NAND9 G Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M11 NAND9 H Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M12 NAND9 I Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M13 NAND9 F Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M14 NAND9 E Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M15 NAND9 D Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M16 NAND9 C Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M17 NAND9 B Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M18 NAND9 A Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_MUX_2x1_ACT C0 Cis0 Cis1 Ov Gnd Vdd
Xcaa_INV_1 C0 nC0 Gnd Vdd caa_INV
Xcaa_INV_2 Cis0 nCis0 Gnd Vdd caa_INV
Xcaa_INV_3 Cis1 nCis1 Gnd Vdd caa_INV
M1 N4 nC0 Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 Ov nCis0 N4 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N10 C0 Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 Ov nCis1 N10 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N3 C0 Vdd Vdd CMOSP W=14*lambda L=2*lambda AS=14*lambda*5.5*lambda
AD=14*lambda*5.5*lambda PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
```

```
M6 Ov nCis0 N3 Vdd CMOSP W=14*lambda L=2*lambda AS=14*lambda*5.5*lambda
AD=14*lambda*5.5*lambda PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M7 N8 nC0 Vdd Vdd CMOSP W=14*lambda L=2*lambda AS=14*lambda*5.5*lambda
AD=14*lambda*5.5*lambda PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
M8 Ov nCis1 N8 Vdd CMOSP W=14*lambda L=2*lambda AS=14*lambda*5.5*lambda
AD=14*lambda*5.5*lambda PS=14*lambda+14*lambda+5.5*lambda+5.5*lambda
PD=14*lambda+14*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_NAND8 A B C D E F G H NAND8 Gnd Vdd
M1 NAND8 A N1 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 N1 B N5 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 N5 C N8 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 N8 D N11 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M5 N11 E N14 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M6 N14 F N17 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M7 N17 G N20 Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M8 N20 H Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M9 NAND8 G Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M10 NAND8 H Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M11 NAND8 F Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M12 NAND8 E Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M13 NAND8 D Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M14 NAND8 C Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
```

```
M15 NAND8 B Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M16 NAND8 A Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_overflow C0 G3 nG0 nG1 nG2 nP3 Ov P0 P1 P2 Gnd Vdd
Xcaa_INV_1 nG0 G0 Gnd Vdd caa_INV
Xcaa_INV_2 P0 nP0 Gnd Vdd caa_INV
Xcaa_INV_3 nG1 G1 Gnd Vdd caa_INV
Xcaa_INV_4 P1 nP1 Gnd Vdd caa_INV
Xcaa_INV_5 nG2 G2 Gnd Vdd caa_INV
Xcaa_INV_6 P2 nP2 Gnd Vdd caa_INV
Xcaa_INV_7 G3 nG3 Gnd Vdd caa_INV
Xcaa_INV_8 nP3 P3 Gnd Vdd caa_INV
Xcaa_MUX_2x1_ACT_1 C0 N3 N2 Ov Gnd Vdd caa_MUX_2x1_ACT
Xcaa_NAND4_1 nG3 G2 G1 G0 N1 Gnd Vdd caa_NAND4
Xcaa_NAND4_2 nG0 P3 nP2 nP1 N10 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 nG3 P2 nP1 nP0 N9 Gnd Vdd caa_NAND4
Xcaa_NAND4_4 nG3 G0 P2 nP1 N8 Gnd Vdd caa_NAND4
Xcaa_NAND4_5 nG3 G2 P1 nP0 N7 Gnd Vdd caa_NAND4
Xcaa_NAND4_6 nG2 G1 P3 nP0 N19 Gnd Vdd caa_NAND4
Xcaa_NAND4_7 nG2 G1 G0 P3 N18 Gnd Vdd caa_NAND4
Xcaa_NAND4_8 nG1 G0 P3 nP2 N17 Gnd Vdd caa_NAND4
Xcaa_NAND4_9 nG3 G2 G1 P0 N15 Gnd Vdd caa_NAND4
Xcaa_NAND4_10 P3 nP2 nP1 nP0 N46 Gnd Vdd caa_NAND4
Xcaa_NAND8_1 N1 N10 N9 N8 N7 N19 N18 N17 N3 Gnd Vdd caa_NAND8
Xcaa_NAND8_2 N9 N8 N7 N19 N18 N17 N15 N46 N2 Gnd Vdd caa_NAND8
.ENDS

.SUBCKT caa_NOR2 A B NOR2 Gnd Vdd
M1 NOR2 A Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 NOR2 B Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 NOR2 B N9 Vdd CMOSP W=17*lambda L=2*lambda AS=17*lambda*5.5*lambda
AD=17*lambda*5.5*lambda PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
M4 N9 A Vdd Vdd CMOSP W=17*lambda L=2*lambda AS=17*lambda*5.5*lambda
AD=17*lambda*5.5*lambda PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_PG_GENx1 A B G nG nP P Gnd Vdd
Xcaa_INV_1 G nG Gnd Vdd caa_INV
Xcaa_INV_2 P nP Gnd Vdd caa_INV
Xcaa_NAND2_1 A B G Gnd Vdd caa_NAND2
Xcaa_NOR2_1 A B P Gnd Vdd caa_NOR2
.ENDS

.SUBCKT caa_PG_GENx4 A0 A1 A2 A3 B0 B1 B2 B3 C0 G0 G1 G2 G3 nC0 nG0 nG1
nG2 nG3 nP0 nP1 nP2 nP4 P0 P1 P2 P3 Gnd Vdd
```

```
Xcaa_INV_1 C0 nC0 Gnd Vdd caa_INV
Xcaa_PG_GENx1_1 A0 B0 G0 nG0 nP0 P0 Gnd Vdd caa_PG_GENx1
Xcaa_PG_GENx1_2 A1 B1 G1 nG1 nP1 P1 Gnd Vdd caa_PG_GENx1
Xcaa_PG_GENx1_3 A2 B2 G2 nG2 nP2 P2 Gnd Vdd caa_PG_GENx1
Xcaa_PG_GENx1_4 A3 B3 G3 nG3 nP4 P3 Gnd Vdd caa_PG_GENx1
.ENDS

.SUBCKT caa_Adderx4 A0 A1 A2 A3 B0 B1 B2 B3 C0 Ov S0 S1 S2 S3 Gnd Vdd
Xcaa_NAND13_1 N45 N44 N43 N42 N40 N38 N37 N39 N41 N46 N47 N48 N49 S3
Gnd Vdd caa_NAND13
Xcaa_NAND2_1 C0 nG0 N20 Gnd Vdd caa_NAND2
Xcaa_NAND2_2 C0 P0 N19 Gnd Vdd caa_NAND2
Xcaa_NAND2_3 nG1 nG0 N27 Gnd Vdd caa_NAND2
Xcaa_NAND2_4 nG0 P1 N26 Gnd Vdd caa_NAND2
Xcaa_NAND2_5 nG2 nG1 N34 Gnd Vdd caa_NAND2
Xcaa_NAND2_6 nG1 P2 N32 Gnd Vdd caa_NAND2
Xcaa_NAND3_1 nC0 G0 nP0 N21 Gnd Vdd caa_NAND3
Xcaa_NAND3_2 C0 nG1 nP0 N23 Gnd Vdd caa_NAND3
Xcaa_NAND3_3 G1 nP1 P0 N24 Gnd Vdd caa_NAND3
Xcaa_NAND3_4 C0 P1 nP0 N25 Gnd Vdd caa_NAND3
Xcaa_NAND3_5 nG2 nG0 nP1 N33 Gnd Vdd caa_NAND3
Xcaa_NAND3_6 G2 nP2 P1 N31 Gnd Vdd caa_NAND3
Xcaa_NAND3_7 G1 nG2 nG3 N49 Gnd Vdd caa_NAND3
Xcaa_NAND3_8 nP0 P3 nG2 N46 Gnd Vdd caa_NAND3
Xcaa_NAND3_9 nP2 P3 nG1 N37 Gnd Vdd caa_NAND3
Xcaa_NAND3_10 nG0 P2 nP1 N29 Gnd Vdd caa_NAND3
Xcaa_NAND3_11 N20 N19 N21 S0 Gnd Vdd caa_NAND3
Xcaa_NAND4_1 nC0 G1 G0 nP1 N22 Gnd Vdd caa_NAND4
Xcaa_NAND4_2 C0 nG2 nP1 nP0 N30 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 G2 G1 nP2 P0 N35 Gnd Vdd caa_NAND4
Xcaa_NAND4_4 C0 P2 nP1 nP0 N36 Gnd Vdd caa_NAND4
Xcaa_NAND4_5 nP2 G0 nG1 nG3 N48 Gnd Vdd caa_NAND4
Xcaa_NAND4_6 nP1 nP2 nG0 nG3 N47 Gnd Vdd caa_NAND4
Xcaa_NAND4_7 G0 P3 G1 nG2 N41 Gnd Vdd caa_NAND4
Xcaa_NAND4_8 nP1 P2 nP3 G3 N39 Gnd Vdd caa_NAND4
Xcaa_NAND4_9 nP3 P1 G2 G3 N38 Gnd Vdd caa_NAND4
Xcaa_NAND5_1 nC0 G2 G1 G0 nP2 N28 Gnd Vdd caa_NAND5
Xcaa_NAND5_2 nP1 nP2 P3 nG0 G3 N40 Gnd Vdd caa_NAND5
Xcaa_NAND5_3 nP0 nP1 nP2 nG3 C0 N43 Gnd Vdd caa_NAND5
Xcaa_NAND5_4 P0 nP3 G1 G2 G3 N44 Gnd Vdd caa_NAND5
Xcaa_NAND5_5 nP0 nP1 nP2 P3 C0 N45 Gnd Vdd caa_NAND5
Xcaa_NAND6_1 nP3 G0 G1 G2 G3 nC0 N42 Gnd Vdd caa_NAND6
Xcaa_NAND6_2 N27 N26 N22 N23 N24 N25 S1 Gnd Vdd caa_NAND6
Xcaa_NAND9_1 N34 N33 N32 N31 N29 N28 N30 N35 N36 S2 Gnd Vdd caa_NAND9
Xcaa_overflow_1 C0 G3 nG0 nG1 nG2 nP3 Ov P0 P1 P2 Gnd Vdd caa_overflow
XPG_GENx4_1 A0 A1 A2 A3 B0 B1 B2 B3 C0 G0 G1 G2 G3 nC0 nG0 nG1 nG2 nG3
nP0 nP1 nP2 nP3 P0 P1 P2 P3 Gnd Vdd caa_PG_GENx4
.ENDS

.SUBCKT caa_NAND15 A B C D E F G H I J K L M N NAND15 O Gnd Vdd
M1 N2 N N1 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M2 N1 O Gnd Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
```

```
M3 N8 M N2 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M4 N11 L N8 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M5 N14 K N11 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M6 N17 J N14 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M7 N20 I N17 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M8 N23 H N20 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M9 N26 G N23 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M10 N29 F N26 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M11 N32 E N29 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M12 N35 D N32 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M13 N38 C N35 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M14 N41 B N38 Gnd CMOSN W=15*lambda L=2*lambda AS=5.5*lambda*15*lambda
AD=5.5*lambda*15*lambda PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
M15 NAND15 A N41 Gnd CMOSN W=15*lambda L=2*lambda
AS=5.5*lambda*15*lambda AD=5.5*lambda*15*lambda
PS=15*lambda+5.5*lambda+15*lambda+5.5*lambda
PD=15*lambda+15*lambda+5.5*lambda+5.5*lambda
*M1 N2 N N1 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M2 N1 O Gnd Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M3 N8 M N2 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M4 N11 L N8 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M5 N14 K N11 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
```

```
*M6 N17 J N14 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M7 N20 I N17 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M8 N23 H N20 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M9 N26 G N23 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M10 N29 F N26 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M11 N32 E N29 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M12 N35 D N32 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M13 N38 C N35 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M14 N41 B N38 Gnd CMOSN W=20*lambda L=2*lambda AS=5.5*lambda*20*lambda
AD=5.5*lambda*20*lambda PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
*M15 NAND15 A N41 Gnd CMOSN W=20*lambda L=2*lambda
AS=5.5*lambda*20*lambda AD=5.5*lambda*20*lambda
PS=20*lambda+5.5*lambda+20*lambda+5.5*lambda
PD=20*lambda+20*lambda+5.5*lambda+5.5*lambda
M16 NAND15 I Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M17 NAND15 H Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M18 NAND15 G Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M19 NAND15 F Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M20 NAND15 E Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M21 NAND15 D Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M22 NAND15 C Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M23 NAND15 B Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
```

```
M24 NAND15 A Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M25 NAND15 J Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M26 NAND15 K Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M27 NAND15 L Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M28 NAND15 M Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M29 NAND15 N Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M30 NAND15 O Vdd Vdd CMOSP W=5*lambda L=2*lambda AS=5*lambda*5.5*lambda
AD=5*lambda*5.5*lambda PS=5*lambda+5*lambda+5.5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_NOR4 A B C D NOR4 Gnd Vdd
M1 NOR4 A Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M2 NOR4 B Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M3 NOR4 C Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
M4 NOR4 D Gnd Gnd CMOSN W=5*lambda L=2*lambda AS=5.5*lambda*5*lambda
AD=5.5*lambda*5*lambda PS=5*lambda+5.5*lambda+5*lambda+5.5*lambda
PD=5*lambda+5*lambda+5.5*lambda+5.5*lambda
*M5 NOR4 B N17 Vdd CMOSP W=25*lambda L=2*lambda AS=25*lambda*5.5*lambda
AD=25*lambda*5.5*lambda PS=25*lambda+25*lambda+5.5*lambda+5.5*lambda
PD=25*lambda+25*lambda+5.5*lambda+5.5*lambda
*M6 N17 A N20 Vdd CMOSP W=25*lambda L=2*lambda AS=25*lambda*5.5*lambda
AD=25*lambda*5.5*lambda PS=25*lambda+25*lambda+5.5*lambda+5.5*lambda
PD=25*lambda+25*lambda+5.5*lambda+5.5*lambda
*M7 N20 C N23 Vdd CMOSP W=25*lambda L=2*lambda AS=25*lambda*5.5*lambda
AD=25*lambda*5.5*lambda PS=25*lambda+25*lambda+5.5*lambda+5.5*lambda
PD=25*lambda+25*lambda+5.5*lambda+5.5*lambda
*M8 N23 D Vdd Vdd CMOSP W=25*lambda L=2*lambda AS=25*lambda*5.5*lambda
AD=25*lambda*5.5*lambda PS=25*lambda+25*lambda+5.5*lambda+5.5*lambda
PD=25*lambda+25*lambda+5.5*lambda+5.5*lambda
M5 NOR4 B N17 Vdd CMOSP W=17*lambda L=2*lambda AS=17*lambda*5.5*lambda
AD=17*lambda*5.5*lambda PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
M6 N17 A N20 Vdd CMOSP W=17*lambda L=2*lambda AS=17*lambda*5.5*lambda
AD=17*lambda*5.5*lambda PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
```

```
M7 N20 C N23 Vdd CMOSP W=17*lambda L=2*lambda AS=17*lambda*5.5*lambda
AD=17*lambda*5.5*lambda PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
M8 N23 D Vdd Vdd CMOSP W=17*lambda L=2*lambda AS=17*lambda*5.5*lambda
AD=17*lambda*5.5*lambda PS=17*lambda+17*lambda+5.5*lambda+5.5*lambda
PD=17*lambda+17*lambda+5.5*lambda+5.5*lambda
.ENDS

.SUBCKT caa_Grp_PG A0 A1 A2 A3 B0 B1 B2 B3 Grp_G Grp_P Gnd Vdd
Xcaa_NAND15_1 N90 N43 N84 N20 N78 N38 N72 N9 N66 N33 N60 N24 N54 N28
Grp_G N48 Gnd Vdd caa_NAND15
Xcaa_NAND2_1 B3 A3 N9 Gnd Vdd caa_NAND2
Xcaa_NAND3_1 B2 A2 A3 N20 Gnd Vdd caa_NAND3
Xcaa_NAND3_2 B2 B3 A2 N24 Gnd Vdd caa_NAND3
Xcaa_NAND4_1 B1 B2 B3 A1 N28 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 B1 B3 A1 A2 N33 Gnd Vdd caa_NAND4
Xcaa_NAND4_4 B1 B2 A1 A3 N38 Gnd Vdd caa_NAND4
Xcaa_NAND4_5 B1 A1 A2 A3 N43 Gnd Vdd caa_NAND4
Xcaa_NAND5_1 B0 B1 B2 B3 A0 N48 Gnd Vdd caa_NAND5
Xcaa_NAND5_2 B0 B2 B3 A0 A1 N54 Gnd Vdd caa_NAND5
Xcaa_NAND5_3 B0 B1 B3 A0 A2 N60 Gnd Vdd caa_NAND5
Xcaa_NAND5_4 B0 B3 A0 A1 A2 N66 Gnd Vdd caa_NAND5
Xcaa_NAND5_5 B0 B1 B2 A0 A3 N72 Gnd Vdd caa_NAND5
Xcaa_NAND5_6 B0 B2 A0 A1 A3 N78 Gnd Vdd caa_NAND5
Xcaa_NAND5_7 B0 B1 A0 A2 A3 N84 Gnd Vdd caa_NAND5
Xcaa_NAND5_8 B0 A0 A1 A2 A3 N90 Gnd Vdd caa_NAND5
Xcaa_NOR2_1 B3 A3 N96 Gnd Vdd caa_NOR2
Xcaa_NOR2_2 B2 A2 N99 Gnd Vdd caa_NOR2
Xcaa_NOR2_3 B1 A1 N102 Gnd Vdd caa_NOR2
Xcaa_NOR2_4 B0 A0 N105 Gnd Vdd caa_NOR2
Xcaa_NOR4_1 N105 N102 N99 N96 Grp_P Gnd Vdd caa_NOR4
.ENDS

.SUBCKT ALUx4 A0 A1 A2 A3 B0 B1 B2 B3 C0 Grp_G Grp_P Ov S0 S1 S2 S3 Gnd
Vdd
Xcaa_Adderx4_1 A0 A1 A2 A3 B0 B1 B2 B3 C0 Ov S0 S1 S2 S3 Gnd Vdd
caa_Adderx4
Xcaa_Grp_PG_1 A0 A1 A2 A3 B0 B1 B2 B3 Grp_G Grp_P Gnd Vdd caa_Grp_PG
.ENDS

* Main circuit: caa_Adderx16
XALUx4_1 A0 A1 A2 A3 B0 B1 B2 B3 C0 N11 N12 N1 S0 S1 S2 S3 Gnd Vdd
ALUx4
XALUx4_2 A4 A5 A6 A7 B4 B5 B6 B7 N8 N9 N10 N2 S4 S5 S6 S7 Gnd Vdd ALUx4
XALUx4_3 A8 A9 A10 A11 B8 B9 B10 B11 N5 N6 N7 N3 S8 S9 S10 S11 Gnd Vdd
ALUx4
XALUx4_4 A12 A13 A14 A15 B12 B13 B14 B15 N14 N15 N4 Ov S12 S13 S14 S15
Gnd Vdd ALUx4
Xcaa_CLAH_1 C0 N8 N5 N14 C16 N11 N9 N6 N15 N12 N10 N7 N4 Gnd Vdd
caa_CLAH
* End of main circuit: caa_Adderx16
.END
```

D. MATLAB CHECK ROUTINE

```
% Addercheck_625.m
% This program will take input from spice transient analysis
% and assign binary values to output at specified (i.e. on the
% clock) times.  It will then match this output to expected
% values.

data = load ('c:\MATLABR11\work\caa_Adder_625.dat');
size(data)

data = data(1:512,:);
time = round(data(:,1)*10^11);
Sdata = data(:,2:17);
Sdata = fliplr(Sdata);
Cdata = data(:,18);
Odata = data(:,19);
Adata = data(:,21:36);
Adata = fliplr(Adata);
Bdata = data(:,37:52);
Bdata = fliplr(Bdata);
Cindata=data(:,53);
Cindata=fliplr(Cindata);

[I,J] = size(time);

% convert voltages to binary values

% 1.  find indices where voltage is high (binary one)
Sone = find(Sdata > 0.9);
Cone = find(Cdata > 0.9);
Oone = find(Odata > 0.9);
Aone = find(Adata > 0.9);
Bone = find(Bdata > 0.9);
Cinone = find(Cindata > 0.9);

% 2.  initialize binary vectors with zero, then set values
corresponding to indices to one

Sbin = zeros(512,16);
Sbin(Sone) = 1;
Cbin = zeros(512,1);
Cbin(Cone) = 1;
Obin = zeros(512,1);
Obin(Oone) = 1;
Abin = zeros(512,16);
Abin(Aone) = 1;
Bbin = zeros(512,16);
Bbin(Bone) = 1;
Cinbin = zeros(512,1);
Cinbin(Cinone) = 1;

% create test matrix of sum bits
CAB = (linspace(0,511,512))';

b = 4;
```

```
for m = 1:512,
   C = bitget(CAB(m),2*b+1);

   for n = 1:b,

      A = bitget(CAB(m),n);
      B = bitget(CAB(m),n+4);
      S(m,n) = xor(xor(A,B),C);
      C = or(or(and(A,B),and(A,C)),and(B,C));
   end

  for n = 5:8,
      A = 1;
      B = 0;
      S(m,n) = xor(xor(A,B),C);
      C = or(or(and(A,B),and(A,C)),and(B,C));
  end
  for n = 9:14,
      A = 1;
      B = 1;
      S(m,n) = xor(xor(A,B),C);
      C = or(or(and(A,B),and(A,C)),and(B,C));
  end
  n = 15;
  B = rem(m,2);
  S(m,n) = xor(xor(A,B),C);
  C = or(or(and(A,B),and(A,C)),and(B,C));

  n = 16;
  A = not(rem(m,2));
  B = not(or(not(rem(m,4)),not(rem(m+1,4))));
  S(m,n) = xor(xor(A,B),C);
  C_out(m) = or(or(and(A,B),and(A,C)),and(B,C));


   ov(m) = xor(C_out(m),C);
end

VS = Sbin - S;
Serror = sum(sum(VS,1))

VO = Obin - ov';
Oerror = sum(sum(VO,1))

VC = Cbin - C_out';
Cerror = sum(sum(VC,1))
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. QUINE-MCCLUSKEY MINIMIZATION ROUTINE

A. OVERVIEW

The following MATLAB routines are used to compute the minimum set of maxterms or minterms for the 4-Bit Adder. Adder.m calculates the G and P variables and calls quineadder.m with the indices of the logic one’s (for minterms) or logic zero’s (for maxterms) and the number of variables. Adder.m leverages the “don’t care” instances as discussed in Figure 10. Quineadder.m in succession calls three subroutines, qneindex.m, qnecmp.m and qneout.m to utilize the Quine-McCluskey tabular routine to determine the minimum minterms or maxterms.

B. ADDER.M

```
diary('Adder_DC_eq_one.txt')
% script file to fill a matrix (S) with a four bit wide addition of
% two 4 bit numbers and carry in.  Once this four bit sum is determined
% the indices of each sum bit can be used as entry into a quine-mclusky
% minimization routine to determine required minterms.

% Build sum matrix.  This matrix is 512 x 4 bits wide
clear;
CAB = (linspace(0,511,512))';

% Modify this script to run for only b bits.  The problem with the
% current script is that the quine-mclusky grinds to a halt after 7 or
% so variables.  Also, there is an efficiency issue.  For S1, the only
% variables are A1, B1 and C1.  S2, is A2, A1, B2, B1, and C1, etc up
% to S4.  The idea is to limit the amount of variables being processed
% to only those necessary in order to save computational time.

for b = 1:4,

   for m = 1:(2^(2*b+1)),
      C = bitget(CAB(m),2*b+1);
      for n = 1:b,
         A = bitget(CAB(m),b+n);
         B = bitget(CAB(m),n);
         S(m,n) = xor(xor(A,B),C);
         C = or(or(and(A,B),and(A,C)),and(B,C));
      end
      S(m,5) = C;
   end
```

```
% find indices for each sum bit and carry out and convert them to the
% decimal representation of the binary number of CAB

   Sind = (find(S(:,b))-1)';   % minterms, add <1 for maxterms

% Take indices and convert them to number representations of
% g and p, use those numbers in quineadder.

   GP_dec= zeros(2^(2*b+1),1);

   for m = 1:(2^(2*b+1)),
      for n = 1:b,
         A = bitget(CAB(m),b+n);
         B = bitget(CAB(m),n);
         GP(m,n) = not(or(A,B)); % calculate Pn
         GP(m,b+n) = not(and(A,B)); % calculate Gn

      end
      GP(m,2*b+1) = bitget(CAB(m),2*b+1);  %set Co
      for i = 1:(2*b+1),
         GP_dec(m) = GP_dec(m) + GP(m,i)*2^(i-1);
      end
   end

% This has been added to force the don't care values of G and P, i.e.
% GP = 01 is don't care, to a value of one for minterms minimization.
   GP_C= sort(GP_dec);
   GP_DC = zeros(2^(2*b+1),1);  % don't care matrix
   nmax = max(GP_C);
   n = 1;
   for m = 0:nmax,
      if ( (m) ~= GP_C(n)),
        GP_DC(m+1) = 1;

      else while (GP_C(n) == (m) & n <= nmax),
            n = n + 1;
            end
      end
   end
   DCind = (find(GP_DC)-1)';   % minterms, add <1 for maxterms

   GP_ind = sort([GP_dec(Sind+1); DCind'])';
% use indices as input arguement to quineadder, a quine-mclusky
% minimization routine.  The output is stored as associated minterms
   b
   GP_DCeqONE_MIN = quineadder(GP_ind, b)

end  %for loop with b as control bit

diary off
```

C. QUINEADDER.M

```
function output=quineadder(X,bits)
global bin index var prm x
%   CALL:  [output]=quineadder(X,bits)
%
%          where:
%          X = row of binary number representations
%              e.g. X=[0,3,4,5,6,7,8,12,13,14,16,21,23,24,29,31];
%          bits = number of variables
q=2*bits + 1;
[bin,index,var,prm]=qneindex(X,q); % call qneindex
n=1;
x=0;
while n<=(q-1)
   a=1;
   m=1;
   small=1;
   while a<=(q+1-n)
      while small<=index(a,n)
         big=index(a,n)+1;
         while big<=index(a+1,n)
            [bin,x]=qnecmp(bin(small,n,:),bin(big,n,:),m,n,q);
            if x==1
               %update var matrix
               j=0;
               k=1;
               totvar=2^(q-1);
               varnew=zeros(totvar,1);
               while k<=(2^(n-1))
                  varnew(j+1,1)=var(small,n,k);
                  varnew(j+2,1)=var(big,n,k);
                  j=j+2;
                  k=k+1;
               end
               varnew=sortrows(reshape(varnew,j,(totvar/j)));
               varnew=reshape(varnew,totvar,1);
               var(m,n+1,:)=varnew;
               l=1;
               y=0;
               strm=num2str(var(m,n+1,:));
               while l<m
                  strl=num2str(var(l,n+1,:));
                  i=strcmp(strm,strl);
                  switch i
                     case 1
                        y=1;
                        prm(small,n)=0;
                        prm(big,n)=0;
                  end
                  l=l+1;
               end
               if y==0
                  %update prm
                  prm(small,n)=0;
                  prm(big,n)=0;
                  prm(m,n+1)=1;
                  m=m+1;
                  %update index
                  b=a;
                  while b<=(q+1-n)
                     index (b,n+1)=index(b,n+1)+1;
                     b=b+1;
                  end
               end
            end
            big=big+1;
         end
         small=small+1;
      end
      a=a+1;
   end
   n=n+1;
end
% calculate the output matrix by using qneout to minimize
[output,prime,term]=qneout(bin,var,prm,X,q);
```

D.   QNEINDEX.M

```
function [bin,index,var,prm]=qneindex(Z,q)
global bin index var prm
%
%    CALL:  [bin,index,var,prm]=qneindex(Z,q)
%
%    INPUT:  Z = vector of decimal logic values for minimization
%            q = number of logic variables
%
%    OUTPUT:  bin = 3D matrix w/sorted binary value (strings) to
%                   minimize
%             index = 2D index matrix w/qty's of variables sorted by
%                     binary 1
%             var = 3D matrix w/sorted variable representations
%                   corresponding to the binary values in 'bin'
%             prm = 2D matrix that keeps track of whether each
%                   succeeding level of 'var' are prime indicants
%                   establish the bin matrix

dim=size(Z);
Z=reshape(Z,dim(1,2),dim(1,1));
a=(dec2bin(Z,q)*1)-48;
b=ones(q,1);
c=a*b;
d=[c,Z];
d=sortrows(d,[1 2]);
e=d(:,2);
e=fliplr(dec2bin(e,q));
bin=char(zeros(dim(1,2),q,q)+48);
bin(:,1,:)=e;

% calculation of the index matrix
p=1;
index=zeros(q+1,q);
```

```
while p<=dim(1,2)
   r=c(p)+1;
   while r<=q+1
      index(r,1)=index(r,1)+1;
      r=r+1;
   end
   p=p+1;
end

% establish the var matrix

var=zeros(dim(1,2),q,(2^(q-1)));
var(:,1,1)=d(:,2);

% establish prm matrix

prm=[ones(dim(1,2),1),zeros(dim(1,2),(q-1))];
```

E. QNECMP.M

```
function [bin,x]=qnecmp(X,Y,m,n,q)
global bin x
%
%   CALL:  [bin,x]=qnecmp(X,Y,m,n,q);
%
%   INPUT:  X=first binary string to compare
%           Y=second binary string to compare
%           m=output row of bin matrix
%           n=input (X,Y) column of bin matrix
%           q=number of variables
%
%   OUTPUT:  bin=binary string matrix
%            x= returns x==1 if the input strings (X,Y) form
%               a prime implicant
%
%

bin(m,n+1,:)=X;
p=1;
x=0;
while p<=q
   i=strcmp(X(1,p),Y(1,p));
   switch i
      case 0
         x=x+1;
         bin(m,n+1,p)='x';
   end
   p=p+1;
end
```

F.   QNEOUT.M

```
function [output,prime,term]=qneout(bin,var,prm,Z,q)
%     CALL:   [output,prime,term]=qneout(bin,var,prm,Z,q)
%
%     INPUT:  bin= 3D matrix w/sorted binary value (strings) to
%                  minimize
%             index = 2D index matrix w/qty's of variables sorted by
%                     binary 1
%             var = 3D matrix w/sorted variable representations
%                   corresponding to the binary values in 'bin'
%             prm = 2D matrix that keeps track of whether each
%                   succeeding level of 'var' are prime indicants
%             Z = vector of decimal logic values for minimization
%             q = number of logic variables
%
%    OUTPUT:  output = 2D matrix with minimized terms
%

% establish prime matrix; this matrix will be used to minimize terms
qtymin=size(Z);
prime=zeros(1,qtymin(1,2));
% establish term matrix; this matrix will have each term expressed in
%    it's binary form.  For example, for variable A, a '1' is 'A', a
%    '0' is 'A'', and a 'x' means A is not used in term.
term=char(zeros(1,q)+48); %
% establish check matrix; the first column will have index number of
%    term the second column of the matrix has the size of the term
%   (i.e. how many minterms it includes).  The third column will be
%   used during the minimization process to iteratively record how
%    many minterms a term is 'currently' including (prioritization).
%    This matrix will be added to beginning of 'prime' and 'term' for
%    sorting.
check=zeros(1,3);

dim=size(prm);
j=1;
n=1;
while n<=dim(1,2)
   m=1;
   while m<=dim(1,1)
      if prm(m,n)==1
         check(j,:)=2^(n-1);  % establish potential # minterms covered
         check(j,1)=j;
         k=1;
         l=1;
         % update prime matrix
         while k<=qtymin(1,2)
            if var(m,n,l)==Z(k)
               prime(j,k)=1;
               l=l+1;
            end
            if l>check(j,2)
               k=qtymin(1,2);   % inserted all minterms into prime
            end
            k=k+1;
         end
         % update term matrix
         term(j,:)=bin(m,n,:);
         j=j+1;
      end
      m=m+1;
   end
   n=n+1;
end
numterm=j-1;
% so ends the establishment of matrixes.  Now, incorporate 'check'
%prm  %print prm for debug
prime=[check,prime];
epi=sum(prime);
r=1;
s=4;
while s<=(qtymin(1,2)+3)
   % update prime
   if epi(1,s)==1
      prime=flipud(sortrows(prime,s));
      if prime(1,s)==1
         t=4;
         while t<=(qtymin(1,2)+3)
            if prime(1,t)==1
               prime(:,t)=0;
            end
            t=t+1;
         end
         % place minimized term into output
         %prime % print for debug
         output(r,:)=term(prime(1,1),:);
         r=r+1;
      end
   end
   s=s+1;
end

while prime(1,3)>0
   % reset second index with remaining terms from minimize
   secind=prime;
   secind(:,1:3)=0;
   secsum=sum(rot90(secind));
   secsum=reshape(secsum,numterm,1);
   prime(:,3)=secsum(:,1);
   prime=flipud(sortrows(prime,[3,2]));
   % is there any remainder?
   if prime(1,3)>0
      % this is a test loop to prioritize next terms for minimize
      % see if there are any terms with only one minterm remaining
      one=find(prime(:,3)==1);
      [I,J]=size(one);
      i=1;
      if I>0
         j=(find(prime(one(1),[4:qtymin(1,2)+3])==1)+3);
         i=find(prime(:,j)==1);
      end
      % minimize remainder with next priority term
      t=4;
      while t<=(qtymin(1,2)+3)
         if prime(i(1),t)==1
            prime(:,t)=0;
         end
         t=t+1;
      end
      %prime % print prime for debug
      % place minimized term into output
      output(r,:)=term(prime(i(1),1),:);
      r=r+1;
   end
end
% place output matrix in variable form, reducing to # variables
output=fliplr(output(:,1:q));
```

APPENDIX E. 4-BIT PSEUDO-NMOS NOR ALU SIMULATION

```
*** SPICE Circuit File of ADDER4 made by LASICKT on 03/29/01 ***

* START OF SPHEAD2.CIR
*CMOS Invertor DC Transfer Characteristics

.INCLUDE t0aeBSIM3.txt

*Power Supplies
VPOWER VDS 0 5.0

* Input Signals
VC0 C0 0 PULSE(5 0 1NS 1NS 1NS 4NS 10NS)
VA0 A0 0 PULSE(5 0 1NS 1NS 1NS 4NS 10NS)
VB0 B0 0 PULSE(5 0 1NS 1NS 1NS 9NS 20NS)
VA1 A1 0 PULSE(5 0 1NS 1NS 1NS 14NS 30NS)
VB1 B1 0 PULSE(5 0 1NS 1NS 1NS 19NS 40NS)
VA2 A2 0 PULSE(5 0 1NS 1NS 1NS 4NS 10NS)
VB2 B2 0 PULSE(5 0 1NS 1NS 1NS 9NS 20NS)
VA3 A3 0 PULSE(5 0 1NS 1NS 1NS 14NS 30NS)
VB3 B3 0 PULSE(5 0 1NS 1NS 1NS 19NS 40NS)
* END OF SPHEAD2.CIR

.SUBCKT NOR5 VDS 0 A B C OUT D E
M2 OUT A 0 0 CMOSN W=6U L=2U
M1 OUT 0 VDS VDS CMOSP W=6U L=2U
M4 OUT C 0 0 CMOSN W=6U L=2U
M6 OUT E 0 0 CMOSN W=6U L=2U
M5 OUT D 0 0 CMOSN W=6U L=2U
M3 OUT B 0 0 CMOSN W=6U L=2U
.ENDS

.SUBCKT NOR9 VDS 0 A B C OUT D E F G H I
M1 OUT 0 VDS VDS CMOSP W=9U L=2U
M2 OUT A 0 0 CMOSN W=9U L=2U
M3 OUT B 0 0 CMOSN W=9U L=2U
M4 OUT C 0 0 CMOSN W=9U L=2U
M5 OUT D 0 0 CMOSN W=9U L=2U
M6 OUT E 0 0 CMOSN W=9U L=2U
M7 OUT F 0 0 CMOSN W=9U L=2U
M8 OUT G 0 0 CMOSN W=9U L=2U
M9 OUT H 0 0 CMOSN W=9U L=2U
M10 OUT I 0 0 CMOSN W=9U L=2U
.ENDS

.SUBCKT S2_C4 0 VDS C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' G2 G2' P2 P2' G3
+ G3' P3 P3' 0 VDS S2 C4
X1 VDS 0 G2 G1' P2 VN9 P1' P0 NOR5
X2 VDS 0 C0 G2 G1' VN8 G0' P2 NOR5
X3 VDS 0 G2' G1 G0' VN7 P2 P1 NOR5
X4 VDS 0 G2' G0 P2 VN6 P1 P0 NOR5
X5 VDS 0 G2' G1' P2' VN5 P1' P0 NOR5
```

```
X6 VDS 0 C0 G2' G1' VN4 G0' P2' NOR5
X7 VDS 0 G2 G1' G0' VN3 P2 P0' NOR5
X8 VDS 0 C0' G2' P2 VN2 P1 P0 NOR5
X9 VDS 0 G2' G1' G0' VN1 P2' P0' NOR5
X10 VDS 0 VN9 VN8 VN7 S2 VN6 VN5 VN4 VN3 VN2 VN1 NOR9
X11 VDS 0 P2 P1 G3' VN18 P3' P0 NOR5
X12 VDS 0 P0 G2' P1' VN17 G1' G3' NOR5
X13 VDS 0 P3' G3' G0' VN16 P2 P1 NOR5
X14 VDS 0 G2' G0' P2' VN15 P1 G3' NOR5
X15 VDS 0 G3' G1' P3' VN14 P2 P0 NOR5
X16 VDS 0 C0 G2' G1' VN13 G0' G3' NOR5
X17 VDS 0 G2' G3' P0 VN12 P2' P1 NOR5
X18 VDS 0 G0' G3' P2 VN11 P3' G1' NOR5
X19 VDS 0 G2' G1' G0' VN10 G3' P0' NOR5
X20 VDS 0 VN18 VN17 VN16 C4 VN15 VN14 VN13 VN12 VN11 VN10 NOR9
.ENDS

.SUBCKT NOR3 VDS 0 A B C OUT
M2 OUT A 0 0 CMOSN W=6U L=2U
M1 OUT 0 VDS VDS CMOSP W=6U L=2U
M4 OUT C 0 0 CMOSN W=6U L=2U
M3 OUT B 0 0 CMOSN W=6U L=2U
.ENDS

.SUBCKT NOR4 VDS 0 A B C OUT D
M2 OUT A 0 0 CMOSN W=6U L=2U
M1 OUT 0 VDS VDS CMOSP W=6U L=2U
M4 OUT C 0 0 CMOSN W=6U L=2U
M5 OUT D 0 0 CMOSN W=6U L=2U
M3 OUT B 0 0 CMOSN W=6U L=2U
.ENDS

.SUBCKT NOR6 VDS 0 A B C OUT D E F
M2 OUT A 0 0 CMOSN W=6U L=2U
M1 OUT 0 VDS VDS CMOSP W=6U L=2U
M4 OUT C 0 0 CMOSN W=6U L=2U
M6 OUT E 0 0 CMOSN W=6U L=2U
M5 OUT D 0 0 CMOSN W=6U L=2U
M7 OUT F 0 0 CMOSN W=6U L=2U
M3 OUT B 0 0 CMOSN W=6U L=2U
.ENDS

.SUBCKT S0_S1 0 VDS S0 C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' S1 0 VDS
X2 VDS 0 C0 G0' P0' VN2 NOR3
X1 VDS 0 C0 G0 P0 VN1 NOR3
X3 VDS 0 C0' G0' P0 VN3 NOR3
X4 VDS 0 VN1 VN2 VN3 S0 NOR3
X6 VDS 0 G1' P1 G0 VN5 P0 NOR4
X7 VDS 0 G1' C0 G0' VN6 P1' NOR4
X8 VDS 0 G1 G0' P1 VN7 P0' NOR4
X9 VDS 0 C0' G1' P1 VN8 P0 NOR4
X10 VDS 0 G1' G0' P1' VN9 P0' NOR4
X5 VDS 0 G0' G0' G1 VN4 C0 NOR4
X11 VDS 0 VN4 VN5 VN6 S1 VN7 VN8 VN9 NOR6
.ENDS
```

```
.SUBCKT NOR15 VDS 0 A B C OUT D E F G H I J K L M N O
M1 OUT 0 VDS VDS CMOSP W=9U L=2U
M2 OUT A 0 0 CMOSN W=9U L=2U
M3 OUT B 0 0 CMOSN W=9U L=2U
M4 OUT C 0 0 CMOSN W=9U L=2U
M5 OUT D 0 0 CMOSN W=9U L=2U
M6 OUT E 0 0 CMOSN W=9U L=2U
M7 OUT F 0 0 CMOSN W=9U L=2U
M8 OUT G 0 0 CMOSN W=9U L=2U
M9 OUT H 0 0 CMOSN W=9U L=2U
M10 OUT I 0 0 CMOSN W=9U L=2U
M11 OUT J 0 0 CMOSN W=9U L=2U
M12 OUT K 0 0 CMOSN W=9U L=2U
M13 OUT L 0 0 CMOSN W=9U L=2U
M14 OUT M 0 0 CMOSN W=9U L=2U
M15 OUT N 0 0 CMOSN W=9U L=2U
M16 OUT O 0 0 CMOSN W=9U L=2U
.ENDS

.SUBCKT S3 0 VDS C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' G2 G2' P2 P2' G3
+ G3' P3 P3' S3 0 VDS
X1 VDS 0 G3 G2' P3 1 P2' P1 G0' NOR6
X2 VDS 0 G3' G1 G0' 2 P3 P2 P1 NOR6
X3 VDS 0 G3 P1' P3 3 G2' G1' P0 NOR6
X4 VDS 0 C0' G3' P3 4 P2 P1 P0 NOR6
X5 VDS 0 G3' G2 G0' 5 P3 P2 G1' NOR6
X6 VDS 0 G3 G2' G1' 6 G0' P3 P0' NOR6
X7 VDS 0 P3 G3 G2' 7 G0' G1' C0 NOR6
X9 VDS 0 G3 G2' P3 15 P2' P1 P0 NOR6
X10 VDS 0 G3' G1' P0 14 P3 P2 G2 NOR6
X11 VDS 0 G3' P1' P3' 13 G2' G1' P0 NOR6
X12 VDS 0 G0 G3' P3 12 P2 P1 P0 NOR6
X13 VDS 0 G3' G2' G0' 11 P3' P2' P1 NOR6
X14 VDS 0 G3' G2' G1' 10 G0' P3' P0' NOR6
X15 VDS 0 P3' G3' G2' 9 P0 P1 P2' NOR6
X8 VDS 0 C0 G0' G1' 8 G2' G3' P3' NOR6
X16 VDS 0 1 2 3 S3 4 5 6 7 8 9 10 11 12 13 14 15 NOR15
.ENDS

.SUBCKT GP1 VDS 0 A B G InvG P InvP
M5 P 0 VDS VDS CMOSP W=6U L=2U
M6 P A 0 0 CMOSN W=9U L=2U
M7 P B 0 0 CMOSN W=9U L=2U
M11 InvP P VDS VDS CMOSP W=6U L=2U
M9 InvG G VDS VDS CMOSP W=6U L=2U
M1 G A VDS VDS CMOSP W=6U L=2U
M3 G B 9 0 CMOSN W=6U L=2U
M4 9 A 0 0 CMOSN W=6U L=2U
M8 InvG G 0 0 CMOSN W=3U L=2U
M10 InvP P 0 0 CMOSN W=3U L=2U
M2 G B VDS VDS CMOSP W=6U L=2U
.ENDS

.SUBCKT PG1 VDS 0 A B G InvG P InvP
M5 P 0 VDS VDS CMOSP W=6U L=2U
M6 P A 0 0 CMOSN W=9U L=2U
```

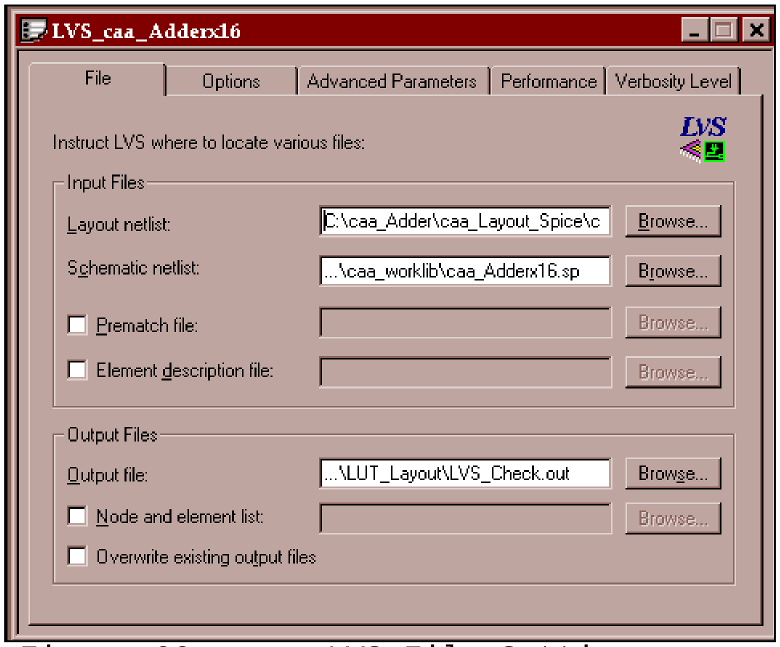
```
M7 P B 0 0 CMOSN W=9U L=2U
M11 InvP P VDS VDS CMOSP W=6U L=2U
M9 InvG G VDS VDS CMOSP W=6U L=2U
M1 G A VDS VDS CMOSP W=6U L=2U
M3 G B 9 0 CMOSN W=6U L=2U
M4 9 A 0 0 CMOSN W=6U L=2U
M8 InvG G 0 0 CMOSN W=3U L=2U
M10 InvP P 0 0 CMOSN W=3U L=2U
M2 G B VDS VDS CMOSP W=6U L=2U
.ENDS

.SUBCKT INV VDS 0 InvIn InvOut
M1 InvOut InvIn 0 0 CMOSN W=3U L=2U
M2 InvOut InvIn VDS VDS CMOSP W=6U L=2U
.ENDS

.SUBCKT PG_GEN 0 VDS C0 A0 B0 A1 B1 A2 B2 A3 B3 C0 C0' G0 G0' P0 P0' G1
+ G1' P1 P1' C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' G2 G2' P2 P2' G3 G3' P3
+ P3' C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' G2 G2' P2 P2' G3 G3' P3 P3'
X1 VDS 0 A0 B0 G0 G0' P0 P0' GP1
X2 VDS 0 B1 A1 G1 G1' P1 P1' PG1
X3 VDS 0 A0 B0 G0 G0' P0 P0' GP1
X4 VDS 0 B1 A1 G1 G1' P1 P1' PG1
X5 VDS 0 A2 B2 G2 G2' P2 P2' GP1
X6 VDS 0 B3 A3 G3 G3' P3 P3' PG1
X7 VDS 0 A0 B0 G0 G0' P0 P0' GP1
X8 VDS 0 B1 A1 G1 G1' P1 P1' PG1
X9 VDS 0 A2 B2 G2 G2' P2 P2' GP1
X10 VDS 0 B3 A3 G3 G3' P3 P3' PG1
X11 VDS 0 C0 C0' INV
.ENDS

*MAIN CIRCUIT ADDER4
X2 0 VDS C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' G2 G2' P2 P2' G3
+ G3' P3 P3' 0 VDS S2 C4 S2_C4
X1 0 VDS S0 C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' S1 0 VDS S0_S1
X3 0 VDS C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' G2 G2' P2 P2' G3
+ G3' P3 P3' S3 0 VDS S3
X4 0 VDS C0 A0 B0 A1 B1 A2 B2 A3 B3 C0 C0' G0 G0' P0 P0' G1 G1'
+ P1 P1' C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' G2 G2' P2 P2' G3
+ G3' P3 P3' C0 C0' G0 G0' P0 P0' G1 G1' P1 P1' G2 G2' P2 P2'
+ G3 G3' P3 P3' PG_GEN

* START OF SPTAIL.CIR

*Simulation Parameters for NFET's
.TRAN 1PS 40NS 0NS

.save V(OUT)
.save i(VPOWER)
.save V(A0)
.save V(B0)

* END OF SPTAIL.CIR
.END
```

APPENDIX F. MOSIS TSMC 0.18 MICRON FET PARAMETERS

```
* MOSIS PARAMETRIC TEST RESULTS
* RUN: T15J (LO_EPI)                         VENDOR: TSMC
* TECHNOLOGY: SCN018                         FEATURE SIZE: 0.18 microns
* T15J SPICE BSIM3 VERSION 3.1 PARAMETERS
* SPICE 3f5 Level 8, Star-HSPICE Level 49, UTMOST Level 8
* DATE: Jul 16/01
* LOT: T15J                  WAF: 5001
* Temperature_parameters=Default

.MODEL CMOSN NMOS (                                LEVEL   = 49
+VERSION = 3.1            TNOM    = 27             TOX     = 4.2E-9
+XJ      = 1E-7           NCH     = 2.3549E17      VTH0    = 0.3593426
+K1      = 0.584235       K2      = 1.808939E-3    K3      = 1E-3
+K3B     = 15.9142604     W0      = 6.767602E-6    NLX     = 1.645593E-7
+DVT0W   = 0              DVT1W   = 0              DVT2W   = 0
+DVT0    = 1.3712712      DVT1    = 0.4653446      DVT2    = -0.0430942
+U0      = 319.668247     UA      = -2.46952E-10   UB      = 6.893182E-19
+UC      = -4.23662E-11   VSAT    = 9.798045E4     A0      = 1.4231374
+AGS     = 0.1896218      B0      = -1.429899E-8   B1      = -1E-7
+KETA    = 0.0270338      A1      = 5.615435E-4    A2      = 0.8500947
+RDSW    = 133.2722527    PRWG    = 0.5            PRWB    = -0.2
+WR      = 1              WINT    = 0              LINT    = 9.682918E-9
+XL      = -2E-8          XW      = -1E-8          DWG     = -7.78854E-9
+DWB     = -1.003184E-8   VOFF    = -0.0652789     NFACTOR = 2.5
+CIT     = 0              CDSC    = 2.4E-4         CDSCD   = 0
+CDSCB   = 0              ETA0    = 0.1006785      ETAB    = -0.0446167
+DSUB    = 0.8210518      PCLM    = 0.7765536      PDIBLC1 = 0.1854406
+PDIBLC2 = 9.865273E-3    PDIBLCB = -0.0540508     DROUT   = 0.8266372
+PSCBE1  = 7.672864E10    PSCBE2  = 2.036021E-8    PVAG    = 0
+DELTA   = 0.01           RSH     = 6.8            MOBMOD  = 1
+PRT     = 0              UTE     = -1.5           KT1     = -0.11
+KT1L    = 0              KT2     = 0.022          UA1     = 4.31E-9
+UB1     = -7.61E-18      UC1     = -5.6E-11       AT      = 3.3E4
+WL      = 0              WLN     = 1              WW      = 0
+WWN     = 1              WWL     = 0              LL      = 0
+LLN     = 1              LW      = 0              LWN     = 1
+LWL     = 0              CAPMOD  = 2              XPART   = 0.5
+CGDO    = 7.23E-10       CGSO    = 7.23E-10       CGBO    = 1E-12
+CJ      = 9.89627E-4     PB      = 0.73534        MJ      = 0.3594267
+CJSW    = 2.46165E-10    PBSW    = 0.7840557      MJSW    = 0.1075765
+CJSWG   = 3.3E-10        PBSWG   = 0.7840557      MJSWG   = 0.1075765
+CF      = 0              PVTH0   = -3.498648E-5   PRDSW   = -2.9489679
+PK2     = -1.251474E-3   WKETA   = 1.928603E-3    LKETA   = -8.378587E-3
+PU0     = 31.1137209     PUA     = 1.155019E-10   PUB     = 0
+PVSAT   = 1.542088E3     PETA0   = -1.003159E-4   PKETA   = 5.130701E-3     )

.MODEL CMOSP PMOS (                                LEVEL   = 49
+VERSION = 3.1            TNOM    = 27             TOX     = 4.2E-9
+XJ      = 1E-7           NCH     = 4.1589E17      VTH0    = -0.4139661
+K1      = 0.5684869      K2      = 0.0351909      K3      = 0
+K3B     = 10.6033883     W0      = 1E-6           NLX     = 9.038631E-8
+DVT0W   = 0              DVT1W   = 0              DVT2W   = 0
+DVT0    = 0.5244177      DVT1    = 0.2901433      DVT2    = 0.1
+U0      = 124.8628741    UA      = 1.792035E-9    UB      = 1E-21
+UC      = -1E-10         VSAT    = 1.551654E5     A0      = 1.5201757
+AGS     = 0.3427925      B0      = 1.666904E-6    B1      = 5E-6
+KETA    = 0.0212022      A1      = 0.028014       A2      = 1
+RDSW    = 304.979313     PRWG    = 0.5            PRWB    = -0.5
+WR      = 1              WINT    = 0              LINT    = 2.053267E-8
+XL      = -2E-8          XW      = -1E-8          DWG     = -3.938518E-8

+DWB     = 5.971841E-9    VOFF    = -0.100662      NFACTOR = 1.9470845
+CIT     = 0              CDSC    = 2.4E-4         CDSCD   = 0
+CDSCB   = 0              ETA0    = 0.2098261      ETAB    = -0.2406335
+DSUB    = 1.2865683      PCLM    = 2.544679       PDIBLC1 = 6.316635E-3
+PDIBLC2 = 0.0508323      PDIBLCB = -9.99311E-4    DROUT   = 0
+PSCBE1  = 1.733444E9     PSCBE2  = 5.00159E-10    PVAG    = 15
+DELTA   = 0.01           RSH     = 7.6            MOBMOD  = 1
+PRT     = 0              UTE     = -1.5           KT1     = -0.11
+KT1L    = 0              KT2     = 0.022          UA1     = 4.31E-9
+UB1     = -7.61E-18      UC1     = -5.6E-11       AT      = 3.3E4
+WL      = 0              WLN     = 1              WW      = 0
+WWN     = 1              WWL     = 0              LL      = 0
+LLN     = 1              LW      = 0              LWN     = 1
+LWL     = 0              CAPMOD  = 2              XPART   = 0.5
+CGDO    = 6.92E-10       CGSO    = 6.92E-10       CGBO    = 1E-12
+CJ      = 1.204978E-3    PB      = 0.8428469      MJ      = 0.4043249
+CJSW    = 2.088728E-10   PBSW    = 0.5832884      MJSW    = 0.3016152
+CJSWG   = 4.22E-10       PBSWG   = 0.5832884      MJSWG   = 0.3016152
+CF      = 0              PVTH0   = 2.844904E-3    PRDSW   = 6.5073202
+PK2     = 2.629498E-3    WKETA   = 2.438155E-3    LKETA   = -4.928775E-3
+PU0     = -2.2589171     PUA     = -7.99545E-11   PUB     = 2.472552E-22
+PVSAT   = -50            PETA0   = 1E-4           PKETA   = 2.018007E-3     )
```

APPENDIX G. LAYOUT VERSUS SCHEMATIC SETUP

The following screen captures detail the proper settings for the LVS check. The GUI comes up when you create a new, or open, a LVS check file.



Instruct LVS where to locate various files:

Input Files

Layout netlist: C:\caa_Adder\caa_Layout_Spice\c

Schematic netlist: ...\caa_worklib\caa_Adderx16.sp

☐ Prematch file:

☐ Element description file:

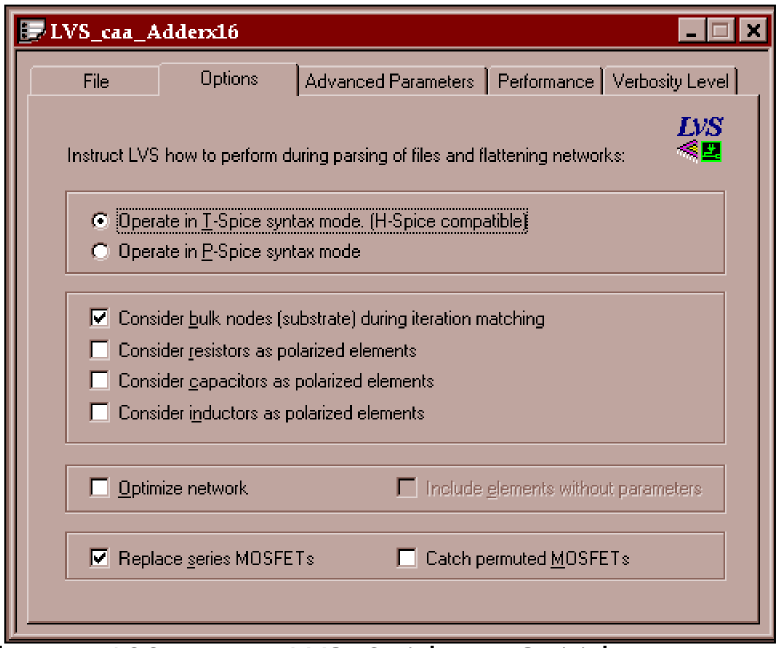
Output Files

Output file: ...\LUT_Layout\LVS_Check.out

☐ Node and element list:

☐ Overwrite existing output files

Figure 99. LVS File Settings.



Instruct LVS how to perform during parsing of files and flattening networks:

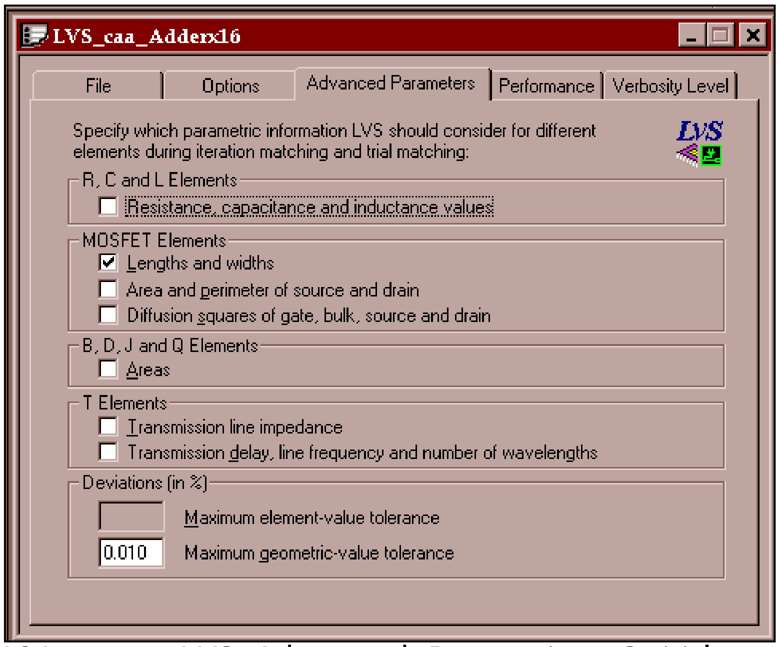
● Operate in T-Spice syntax mode. (H-Spice compatible)
○ Operate in P-Spice syntax mode

☑ Consider bulk nodes (substrate) during iteration matching
☐ Consider resistors as polarized elements
☐ Consider capacitors as polarized elements
☐ Consider inductors as polarized elements

☐ Optimize network ☐ Include elements without parameters

☑ Replace series MOSFETs ☐ Catch permuted MOSFETs

Figure 100. LVS Options Settings.



Specify which parametric information LVS should consider for different elements during iteration matching and trial matching:

R, C and L Elements
☐ Resistance, capacitance and inductance values

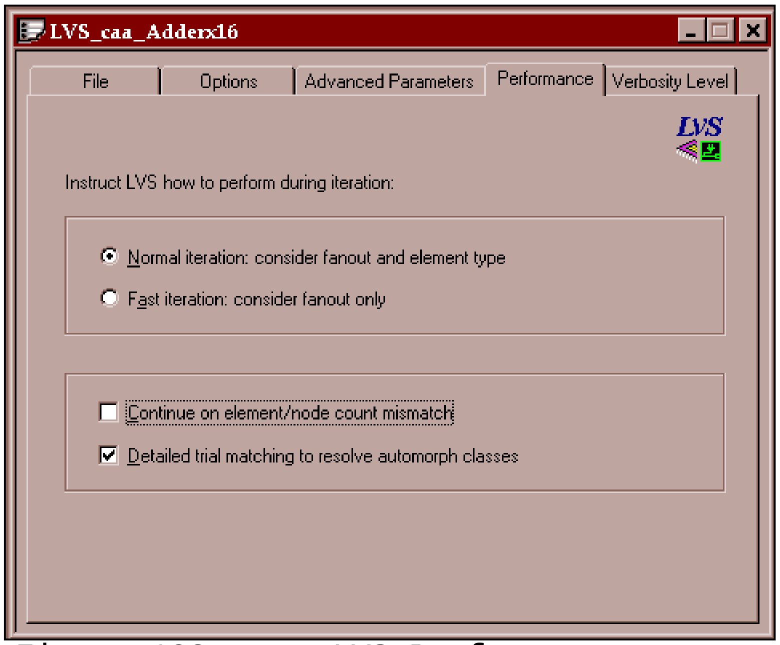
MOSFET Elements
☑ Lengths and widths
☐ Area and perimeter of source and drain
☐ Diffusion squares of gate, bulk, source and drain

B, D, J and Q Elements
☐ Areas

T Elements
☐ Transmission line impedance
☐ Transmission delay, line frequency and number of wavelengths

Deviations (in %)
Maximum element-value tolerance
0.010 Maximum geometric-value tolerance

Figure 101. LVS Advanced Parameter Settings.


LVS_caa_Adderx16

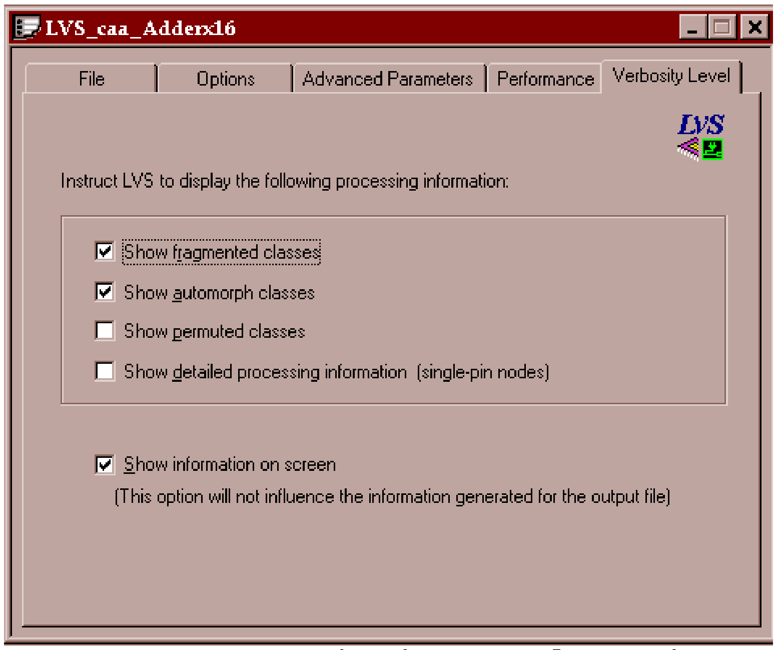
File | Options | Advanced Parameters | Performance | Verbosity Level

Instruct LVS how to perform during iteration:

- (●) Normal iteration: consider fanout and element type
- ( ) Fast iteration: consider fanout only

- [ ] Continue on element/node count mismatch
- [✓] Detailed trial matching to resolve automorph classes

Figure 102. LVS Performance.


LVS_caa_Adderx16

File | Options | Advanced Parameters | Performance | Verbosity Level

Instruct LVS to display the following processing information:

- [✓] Show fragmented classes
- [✓] Show automorph classes
- [ ] Show permuted classes
- [ ] Show detailed processing information (single-pin nodes)

- [✓] Show information on screen
(This option will not influence the information generated for the output file)

Figure 103. LVS Verbosity Level Settings.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. RECOMMENDATIONS FOR FUTURE WORK

SPICE simulations were conducted with the 16-Bit Adder implementing a 4-Bit Adder with sum outputs MUX with C_0 (Figure 104) and a 16-bit overflow circuit (Figure 105), vice overflow circuits on each 4-Bit ALU. The 16-Bit Adder was tested within the pipelined register architecture of the DIS (Figure 106). This is the most realistic simulation possible for the modified adder. The modified adder design operated with a 625 MHz clock speed.

A. 16-BIT OVERFLOW CIRCUIT VERSUS 4-BIT OVERFLOW

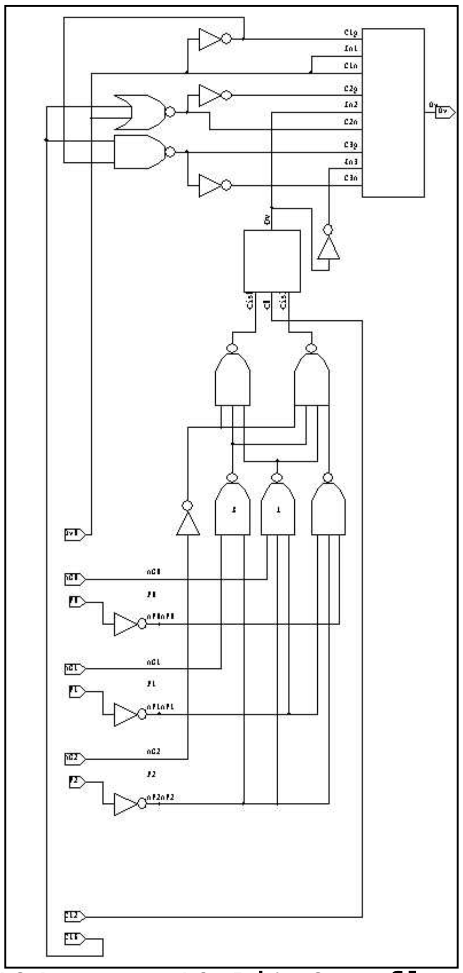


Figure 104. 16-Bit Overflow.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Sep 20, 2001 at 09:42:55
* Main circuit: caa_overflow_x16_3
Xcaa_INV_1 N4 N14 Gnd Vdd caa_INV
Xcaa_INV_2 P0 nP0 Gnd Vdd caa_INV
Xcaa_INV_3 nG2 N11 Gnd Vdd caa_INV
Xcaa_INV_4 P1 nP1 Gnd Vdd caa_INV
Xcaa_INV_5 N16 N1 Gnd Vdd caa_INV
Xcaa_INV_6 P2 nP2 Gnd Vdd caa_INV
Xcaa_INV_7 N5 N13 Gnd Vdd caa_INV
Xcaa_INV_8 Ov0 N3 Gnd Vdd caa_INV
Xcaa_MUX_2x1_ACT_1 C12 N23 N22 N16 Gnd Vdd caa_MUX_2x1_ACT
Xcaa_MUX_3x1_INACT_1 Ov0 N3 N5 N13 N14 N4 Ov0 N16 N1 Ov Gnd Vdd
+ caa_MUX_3x1_INACT
Xcaa_NAND2_1 nG1 nP2 N10 Gnd Vdd caa_NAND2
Xcaa_NAND2_2 C16 N3 N4 Gnd Vdd caa_NAND2
Xcaa_NAND3_1 nG0 nP2 nP1 N12 Gnd Vdd caa_NAND3
Xcaa_NAND3_2 N11 N10 N12 N23 Gnd Vdd caa_NAND3
Xcaa_NAND3_3 nP1 nP2 nP0 N6 Gnd Vdd caa_NAND3
Xcaa_NAND4_1 N11 N10 N12 N6 N22 Gnd Vdd caa_NAND4
Xcaa_NOR2_1 C16 Ov0 N5 Gnd Vdd caa_NOR2
* End of main circuit: caa_overflow_x16_3
.END
```

B. MULTIPLEXING SUM CIRCUITS WITH C_0

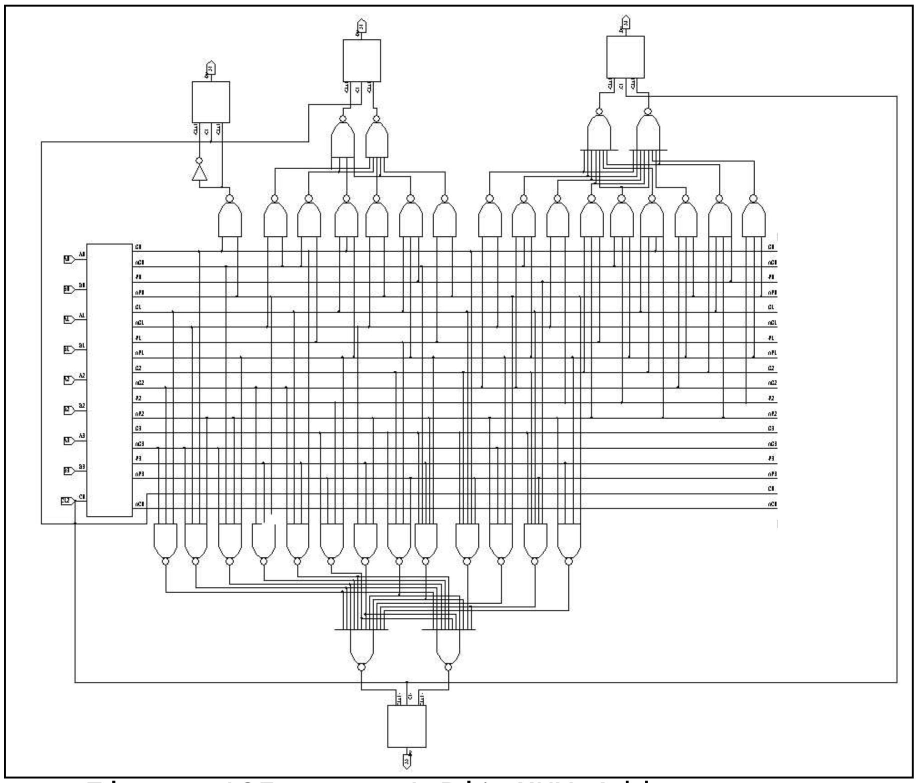


Figure 105. 4-Bit MUX Adder.

```
* SPICE netlist written by S-Edit Win32 7.00
* Written on Sep 20, 2001 at 00:05:45
* Main circuit: caa_Adderx4_MUX_NoOv
Xcaa_INV_2 N10 N12 Gnd Vdd caa_INV
Xcaa_MUX_2x1_ACT_1 C0 N56 N55 S3 Gnd Vdd caa_MUX_2x1_ACT
Xcaa_MUX_2x1_ACT_2 C0 N13 N14 S2 Gnd Vdd caa_MUX_2x1_ACT
Xcaa_MUX_2x1_ACT_3 C0 N12 N10 S0 Gnd Vdd caa_MUX_2x1_ACT
Xcaa_MUX_2x1_ACT_4 C0 N2 N1 S1 Gnd Vdd caa_MUX_2x1_ACT
Xcaa_NAND11_1 N46 N54 N47 N19 N18 N17 N45 N50 N51 N52 N53 N56 Gnd Vdd
+ caa_NAND11
Xcaa_NAND12_1 N48 N46 N49 N47 N19 N18 N17 N45 N50 N51 N52 N53 N55 Gnd
+ Vdd caa_NAND12
Xcaa_NAND2_2 G0 nP0 N10 Gnd Vdd caa_NAND2
Xcaa_NAND2_3 nG1 nG0 N6 Gnd Vdd caa_NAND2
Xcaa_NAND2_4 nG0 P1 N5 Gnd Vdd caa_NAND2
Xcaa_NAND2_5 nG2 nG1 N22 Gnd Vdd caa_NAND2
Xcaa_NAND2_6 nG1 P2 N24 Gnd Vdd caa_NAND2
Xcaa_NAND2_7 nG1 nP0 N7 Gnd Vdd caa_NAND2
Xcaa_NAND2_8 P1 nP0 N3 Gnd Vdd caa_NAND2
Xcaa_NAND3_2 G1 G0 nP1 N8 Gnd Vdd caa_NAND3
Xcaa_NAND3_3 G1 nP1 P0 N4 Gnd Vdd caa_NAND3
Xcaa_NAND3_5 nG2 nG0 nP1 N23 Gnd Vdd caa_NAND3
Xcaa_NAND3_6 G2 nP2 P1 N25 Gnd Vdd caa_NAND3
Xcaa_NAND3_7 G1 nG2 nG3 N53 Gnd Vdd caa_NAND3
Xcaa_NAND3_8 nP0 P3 nG2 N50 Gnd Vdd caa_NAND3
Xcaa_NAND3_9 nP2 P3 nG1 N18 Gnd Vdd caa_NAND3
Xcaa_NAND3_10 nG0 P2 nP1 N16 Gnd Vdd caa_NAND3
Xcaa_NAND3_12 nG2 nP1 nP0 N26 Gnd Vdd caa_NAND3
Xcaa_NAND3_13 P2 nP1 nP0 N20 Gnd Vdd caa_NAND3
Xcaa_NAND4_1 N6 N5 N8 N4 N2 Gnd Vdd caa_NAND4
Xcaa_NAND4_3 G2 G1 nP2 P0 N21 Gnd Vdd caa_NAND4
Xcaa_NAND4_5 nP2 G0 nG1 nG3 N52 Gnd Vdd caa_NAND4
Xcaa_NAND4_6 nP1 nP2 nG0 nG3 N51 Gnd Vdd caa_NAND4
Xcaa_NAND4_7 G0 P3 G1 nG2 N45 Gnd Vdd caa_NAND4
Xcaa_NAND4_8 nP1 P2 nP3 G3 N17 Gnd Vdd caa_NAND4
Xcaa_NAND4_9 nP3 P1 G2 G3 N19 Gnd Vdd caa_NAND4
Xcaa_NAND4_10 nP0 nP1 nG3 nP2 N49 Gnd Vdd caa_NAND4
Xcaa_NAND4_11 nP0 nP1 P3 nP2 N48 Gnd Vdd caa_NAND4
Xcaa_NAND4_12 G1 G2 G0 nP2 N15 Gnd Vdd caa_NAND4
Xcaa_NAND5_1 N6 N5 N7 N4 N3 N1 Gnd Vdd caa_NAND5
Xcaa_NAND5_2 nP1 nP2 P3 nG0 G3 N47 Gnd Vdd caa_NAND5
Xcaa_NAND5_4 P0 nP3 G1 G2 G3 N46 Gnd Vdd caa_NAND5
Xcaa_NAND5_6 nP3 G0 G1 G2 G3 N54 Gnd Vdd caa_NAND5
Xcaa_NAND7_1 N22 N23 N24 N25 N16 N15 N21 N13 Gnd Vdd caa_NAND7
Xcaa_NAND8_1 N22 N23 N24 N25 N16 N20 N26 N21 N14 Gnd Vdd caa_NAND8
XPG_GENx4_1 A0 A1 A2 A3 B0 B1 B2 B3 C0 G0 G1 G2 G3 nC0 nG0 nG1 nG2 nG3
nP0 nP1 nP2 nP3 P0 P1 P2 P3 Gnd Vdd caa_PG_GENx4
* End of main circuit: caa_Adderx4_MUX_NoOv
.END
```

C. PIPELINED REGISTER TEST CIRCUIT

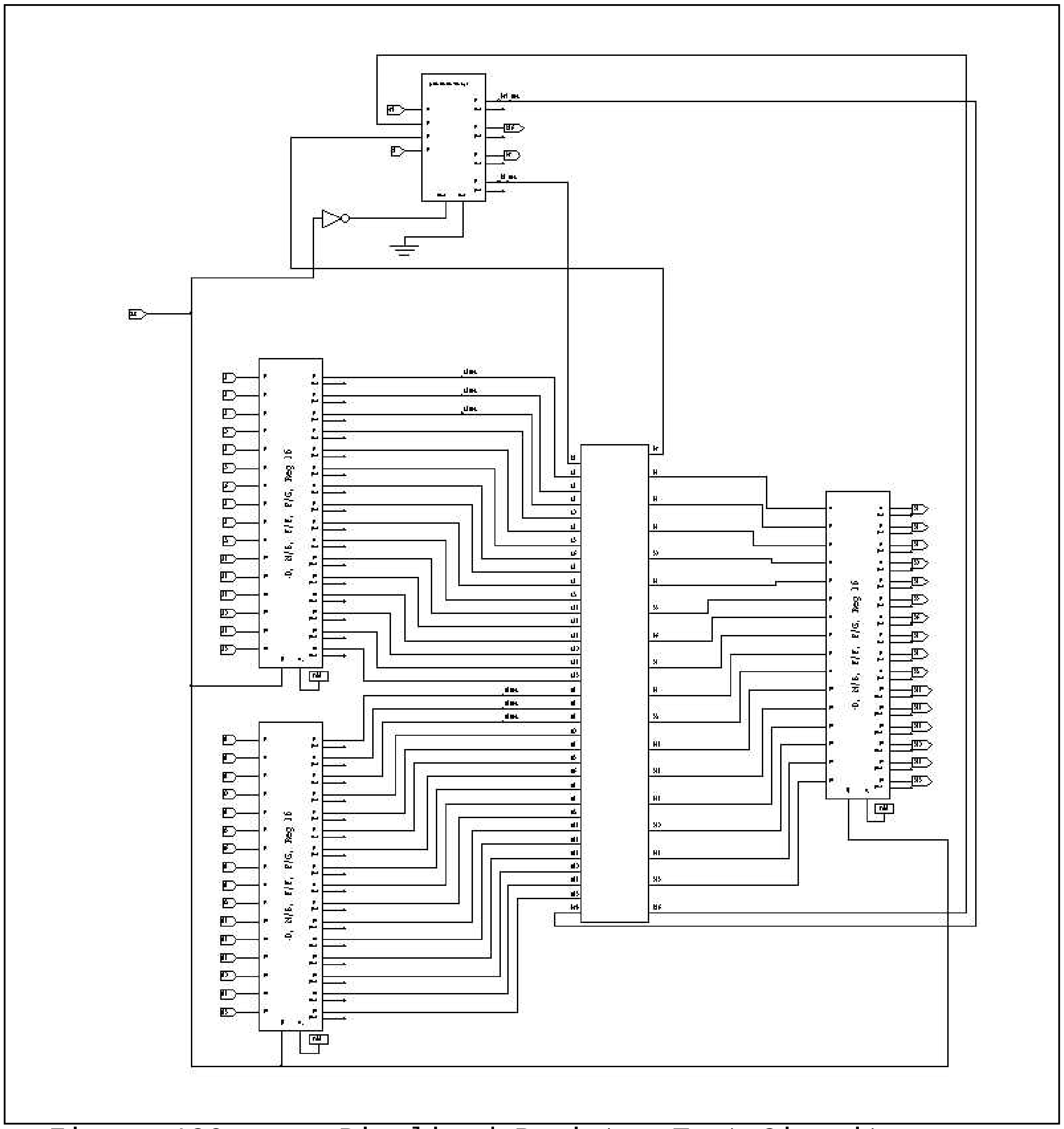


Figure 106. Pipelined Register Test Circuit.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California
   debarber@nps.navy.mil

4. Director, Training and Education, MCCDC, Code C46
   Quantico, Virginia
   webmaster@tecom.usmc.mil

5. Director, Marine Corps Research Center, MCCDC, Code C40RC
   Quantico, Virginia
   ramkeyce@tecom.usmc.mil
   strongka@tecom.usmc.mil
   sanftlebenka@tecom.usmc.mil

6. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
   Camp Pendleton, California
   doranfv@mctssa.usmc.mil
   palanaj@mctssa.usmc.mil

7. Dr. Douglas Fouts, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121
   fouts@nps.navy.mil

8. Dr. Phillip Pace, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, CA 93943-5121
   pepace@nps.navy.mil

9. Naval Research Laboratory
   Code 5740, Bldg 210
   4555 Overlook Ave.
   Washington, D.C. 20375
   ghrin@drsews.nrl.navy.mil
   dbay@drsews.nrl.navy.mil
   cgrounds@drsews.nrl.navy.mil
   Knowles@drsews.nrl.navy.mil

10. Dr. Joe Lawrence, Code ONR-31
Office of Naval Research
Ballston Centre Tower One
800 North Quincy St.
Arlington, VA 22217-5660
Lawrenj@onr.navy.mil
craigp@onr.navy.mil